Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Többmagos mikrokontroller IPC teljesítmény vizsgálata

TDK dolgozat

Készítette:

Kovács Gergely

Konzulens:

dr. Kovácsházy Tamás

2023

Tartalomjegyzék

Kivonat

A dolgozatomban többmagos mikrokontrollerek IPC (Inter-Processor Communication, processzorok vagy folyamatok közötti kommunikáció) teljesítményének vizsgálatával foglalkoztam.

A többmagos mikrokontrollerek és heterogén architektúrák népszerűsége napjainkban növekszik, ugyanis számos feladat esetén előnyös a használatuk. A beágyazott rendszerekben a feladatok száma, valamint számítási igénye folyamatosan növekszik, miközben az energiahatékonyság szintén rendkívül fontos. A processzor növekvő számítási teljesítménye kihasználható, ha a feladatokat a magok között szétosztva párhuzamosan futtatni tudjuk.

Beágyazott operációs rendszereket gyakran alkalmazunk a hatékonyabb és kényelmesebb fejlesztés érdekében. A rendszerhez új feladatok hozzáadása és a meglévő feladatok módosítása egyszerű és rugalmas. Az operációs rendszerek változó mértékben támogatják a többmagos mikrovezérlőket, útmutatást nyújtva a magok közötti kommunikáció megvalósításához.

A kommunikáció vizsgálatával meggyőződhetünk a kiválasztott architektúra megfelelőségéről az adott alkalmazáshoz, valamint kiválaszthatjuk az optimális paramétereket. A mérések elvégzése után levonhatunk általánosan érvényes következtetéseket és megalapozottan hozhatunk döntést az alkalmazás optimalizálásához.

A méréseket az STM32H745 mikrovezérlő használatával végeztem el. Ez egy heterogén architektúrájú kontroller, egy ARM Cortex-M7-es és egy ARM Cortex-M4-es magot tartalmaz. Ezeken külön-külön FreeRTOS beágyazott operációs rendszer futott, amely az IPC-re egy MessageBuffer nevű objektum segítségével ad lehetőséget. Ezen keresztül diszkrét, változtatható méretű üzenetek küldésére van lehetőség. A kommunikáció megosztott memórián keresztül történik, az üzenetek másolásával küldéskor és fogadáskor is.

A mérések során megvizsgáltam több paraméter hatását, például a rendszerben található memóriák közötti különbséget, gyorsítótár használatát, vagy a küldött adat méretének változtatását.

A kommunikáció vizsgálatához létrehoztam egy mérési elrendezést, az operációs rendszer alatt futott. A mérést a PC oldalán egy Python script vezérli soros porton keresztül, ilyenkor konfigurálva a vizsgált paramétereket. A mérési adatok feldolgozása és kiértékelése is itt történik. Az operációs rendszer dokumentációja alapján megvalósítottam a kommunikációt. Az időmérés hardveres számlálók segítségével történik. Az eredmények értékelésekor a vizsgálandó paraméterek hatását tanulmányoztam a kommunikáció adatátviteli sebességére és késleltetésére. Az ábrázoláshoz és az eredmények pontosabb értékeléséhez a mért adatokra lineáris modellt illesztettem, ami a kommunikáció teljesítményét jellemzi.

Abstract

This thesis explores the IPC (Inter-Processor Communication) performance of multi-core microcontrollers.

As the prominence of multi-core microcontrollers rises, their utility proves advantageous for numerous tasks. In embedded systems, the number of tasks and the computational demands are continually increasing, while energy efficiency remains critically important. By parallelizing tasks and distributing them across cores, the computational performance of the processor can be optimized.

Embedded operating systems are often used to facilitate more efficient and streamlined development. They simplify the addition and modification of tasks, which is especially important when working with a high-performance hardware platform. In addition, there is support for multi-core microcontrollers, albeit to varying degrees, and guidance is usually also provided on implementing inter-core communication.

Through an analysis of the communication, we can be convinced of the suitability of the chosen platform for the given application and determine the optimal parameters. After performing the measurements, we can draw broadly applicable conclusions and make informed decisions to optimize the application.

I conducted the measurements using the STM32H745 microcontroller, a device with a heterogeneous architecture, that integrates both an ARM Cortex-M7 and an ARM Cortex-M4 core. Each core operated its own instance of the FreeRTOS operating system. FreeRTOS allows the inter-processor communication using the MessageBuffer communications primitive. The MessageBuffers enable the transmission of discrete messages of varying lengths. Communication occurs via shared memory, involving the copying of messages to and from a shared buffer during both transmission and reception.

To be able to assess the performance of the communication, I established a measurement setup with the help of the aforementioned operating system. Measurements are controlled via serial port via a Python script, which also allows paramterer configuration. Data processing and subsequent evaluation is also performed on the PC side. Based on the operating system's documentation, I implemented the IPC. Runtime evaluation was carried out utilizing hardware counters.

I examined the influence of several parameters, such as the difference between the available integrated memories in the system, implications of cache utilization and the consequences of changing the size of transmitted messages on the data rate and latency of the communication. For improved visualization and more accurate evaluation I fitted a linear model to the measured data points, which characterizes the performance of the communication.

1. fejezet

Bevezetés

Beágyazott rendszerekben a PC-s környezethez hasonlóan a nagy számítási kapacitás és új funkciók iránti igény folyamatosan növekszik, ezért szükség van párhuzamos feldolgozásra. Amdahl törvénye szerint a processzorok számának növelésével felgyorsítható a számítás, ha a megoldandó feladat párhuzamosítható [1]. További előny, hogy a feladatok szeparációjával biztonságosabb alkalmazásokat lehet kialakítani. Ha a független részek külön processzoron hajtódnak végre, akkor az erőforrásokat az egyik CPU-hoz rendelhetjük, a szükséges memóriát és a perifériákat kizárólag egy egységről tesszük elérhetővé. A felek közötti IPC-n kívül nincs kapcsolat, az elválasztott feladatok egymás ütemezésére nincsenek negatív hatással.

A többmagos mikrovezérlőknek számos lehetséges felhasználása merül fel a modern alkalmazások kialakításakor. Leggyakrabban beágyazott rendszerekben egyidejűleg kell számításigényes és determinisztikus időzítést igénylő feladatokat futtatni. Mindezt alacsony késleltetéssel és energiahatékonyan szeretnénk elvégezni.

1.1. Többmagos mikrokontrollerek felhasználási területei

A környezettel szorosan kapcsolatban álló beágyazott rendszerekben az érzékelt jelek feldolgozása gyakran számításigényes feladat [2, 3, 4]. Ezeken a jeleken nagy mintavételi frekvencia mellett kell jelfeldolgozást végezni (például RADAR, LIDAR), vagy akár képek esetén lassabb mintavétel esetén is nagy mennyiségű adat feldolgozása szükséges. A bemenetekből komplex jellemzők kinyerése indokolttá teszi a mesterséges intelligencia alapú, vagy big-data algoritmusok felhasználását beágyazott rendszerekben. A felhasználói interakciók kezeléséhez grafikus felületeket nagy felbontásban kell kezelni. Nagy teljesítményű processzorra van akkor is szükség, ha nagy sebességű kommunikációt szeretnénk folytatni.

A számításokat gyakran az érzékelés után közvetlenül, tehát nem a felhőben kell elvégezni. Erre szükség lehet az energiahatékonyság miatt, ugyanis a nagy mennyiségű adat továbbítása a fogyasztást is növeli. Egy távoli feldolgozás során nehezen teljesíthető követelmény a szükséges kis kisleltetés, ha kommunikáció túl hosszú időt venne igénybe. A feldolgozott adatok lehetnek érzékenyek, amikor is ezek védelme miatt nem megengedett ezeket hálózaton továbbítani.

Az érzékelés és az érzékelt adatok feldolgozása után gyakran a kinyert jellemzők alapján szabályozást végzünk, beavatkozókat vezérlünk és monitorozási feladatokat futtatunk. Ehhez szigorú, valós idejű ütemezésre van szükség.

1.2. Szoftverkörnyezet

Kis teljesítményű mikrovezérlőkben egyszerű feladatok esetén bare metal kódot használunk. A programban általában Round-Robin szoftverarchitektúrát alkalmazunk, vagyis lekérdezést, vagy megszakítást felhasználva perifériákra várakozunk, ha pedig az adatok elérhetőek, akkor végrehajtjuk a feldolgozó kódokat. Ennek a módszernek finomítása a függvénysoros architektúra, amikor egy várakozási sorba helyezünk el futásra kész feldolgozó függvényeket, amiket sorban ütemezünk. Az alkalmazáshoz további feladatok hozzáadása és az ütemezés rugalmatlan.

A komplexebb alkalmazások esetén célszerűbb a nagy számú, eltérő prioritásokkal rendelkező feladatokat beágyazott operációs rendszer (RTOS, real-time operating system) felhasználásával futtatni. A beágyazott operációs rendszerek alkalmazásakor általában preemptív ütemezést használunk, ilyenkor a magasabb prioritású taszkok megszakíthatják az alacsonyabb prioritásúak futását, így kisebb válaszidőt elérve a fontos funkciókra. A beágyazott operációs rendszerek valós idejű ütemezésre képesek, tehát a taszkjaink hívása garantáltan adott határokon belül megtörténik. A fejlesztés során taszkokat kell implementálni, valamint konfigurálni, amit az operációs rendszer ütemez. Ezekben az OS által nyújtott szolgáltatásokat igénybe tudjuk venni, mint a taszkok közötti kommunikáció, vagy akár hardver absztrakciós réteg. Ez sokkal gyorsabb és rugalmasabb fejlesztést tesz lehetővé.

A többmagos mikrokontrollerek esetén a bare metal kód és a beágyazott operációs rendszerek felhasználása lehetséges. Minden mag esetén eldöntendő, hogy melyiket alkalmazzuk, vegyes elrendezésre is van lehetőség. Az operációs rendszer a komplex alkalmazás esetén előnyös, azonban, ha egy mag kis számú algoritmust futtat, akkor érdemes lehet az operációs rendszer futtatásához szükséges számítási többletteljesítményt mellőzni és bare metal megoldással élni.

Habár mikrokontrollereken ez nem lehetséges, de az alkalmazások bonyolultságának további növelésével felmerülhet az igény összetettebb operációs rendszerek felhasználására is. Ilyenkor általában valamilyen asztali környezetben megszokott operációs rendszert futtatunk nagy teljesítményű processzormagokon, általában Linuxot. A szigorú időzítéseket így nehezebb biztosítani, viszont előnyös az OS által biztosított magas absztrakciós szint és az elérhető szabványos megoldások, például a folyamatok közötti kommunikáció megvalósítására.

1.3. Heterogén architektúrák

A többmagos architektúrák hagyományosan azonos processzormagokat tartalmaznak, amelyek között megosztott memória található. Azonban összetett feladatok változatos igényeinek kielégítésére megjelentek a specializált végrehajtóegységek. Így a rendszer egyszerre tud egymásnak ellentmondó követelményeknek is megfelelni, például nagy számítási kapacitást és alacsony fogyasztást biztosítani. Egymás mellett helyezkednek el a rendszerben kis és nagy teljesítményű processzormagok, valamint a speciális számításokra használható hardveres gyorsítók. Ezzel az elrendezéssel az alkalmazás minden komponensét a leghatékonyabb végrehajtóegységen futtathatjuk, így sokkal kedvezőbb teljesítményt és fogyasztást elérve.

Ezeket az elrendezéseket heterogén és homogén multiprocesszoros rendszereknek nevezzük [4]. Homogén esetben a rendszerben található több azonos architektúrájú és mikroarchitektúrájú processzor. Heterogén többmagos rendszerről akkor beszélünk, amikor a magok architektúrája vagy mikroarchitektúrája nem egyezik meg.

Egy processzor architektúrája a funkcionalitást leíró interfészt jelenti, megadja például az utasításkészletet és ezek hatását [5]. A mikroarchitektúra ennek az interfésznek

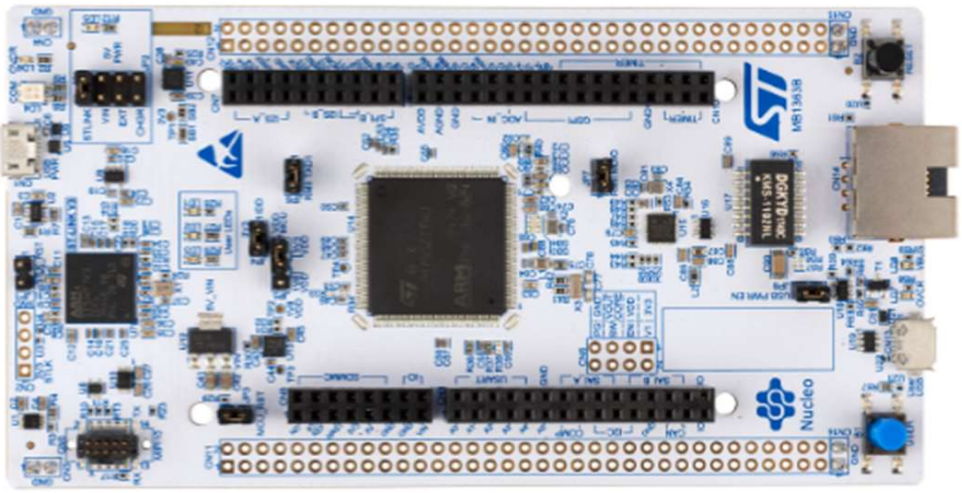
az implementációja, ami a processzor belső működését és felépítését írja le, aminek segítségével az architektúra szerinti interfészt biztosítani tudja. Két processzor, ami azonos architektúrájú, kompatibilis az architektúrához készített szoftverrel, de lehetséges, hogy teljesen eltérő célra vannak kialakítva, például a lehető legjobb teljesítményre, vagy energiahatékonyságra optimalizáltak.

A hardveren elhelyezkedő memóriák megosztottak a processzorok között, a hozzáférési idő lehet független a processzor és a memória elhelyezkedésétől (UMA, uniform memory architecture/access), vagy függhet a processzortól (NUMA, non-uniform memory architecture/access), ilyenkor a lokális memória elérése minden CPU esetén gyorsabb.

Ezzel szemben az AMP (asymmetric multi-processing, aszimmetrikus feladatelosztású feldolgozás) és az SMP (symmetric multi-processing, szimmetrikus feladatelosztású feldolgozás) nem a hardveres felépítést, hanem a rendszer szoftveres környezetét jellemzi. SMP esetén több processzoron fut ugyanaz a kernel példány, tehát egy operációs rendszer felügyeli ezeket a processzorokat. Lehetséges, hogy ezek a CPU-k eltérő architektúrájúak, azonban az ütemezés akkor egyszerűen megvalósítható, ha ezek a CPU-k azonos utasításkészletűek, így a taszkok közöttük dinamikusan szétoszthatóak a szabad erőforrások szerint. A magok között megosztott memóriának kell elhelyezkednie. AMP esetén ezzel szemben a processzorok külön operációs rendszert futtatnak, külön memóriával is rendelkezhetnek. Ezek a processzorok lehetnek azonos architektúrájúak is [6]. A taszkok általában egy adott maghoz vagy operációs rendszerhez vannak rendelve, és nem hajthatóak végre máshol elérhető erőforrás esetén sem. A magok függetlenül kezelhetőek, hasonlóan az egymagos alapesethez, ami megkönnyíti a korábban implementált, nem többmagos kód adaptálását.

Természetesen itt is keverhető egy rendszeren belül a szimmetrikus és aszimmetrikus feladatelosztás, például több azonos processzormag esetén egyetlen asztali OS alkalmazása, emellett pedig mikrokontroller magokon RTOS futtatása.

Heterogén architektúrájú eszközök például az AM243x mikrokontrollerek, amiben ARM Cortex-R és ARM Cortex-M magok találhatóak, valamint az STM32H7x5 mikrokontrollerek, amiben megosztott memóriával két ARM Cortex-M energiahatékony mag található.



1.1. ábra. Az NUCLEO-H745ZI-Q fejlesztői kártya [7]

1.4. Motiváció

Habár ismert, hogy az alkalmazás párhuzamos végrehajtásával az elérhető teljesítmény növelhető, azonban érdemes megvizsgálni, hogy mi a feladatok megosztásával járó többletterhelés. A processzormagok közötti kommunikáció sávszélessége véges és további késleltetést jelent. Ennek a mértéke befolyásolhatja az adott architektúra alkalmazhatóságát.

A szoftver kialakítása során AMP alkalmazásakor többmagos mikrokontrolleren meg kell vizsgálni a kommunikáció teljesítményét, hogy megalapozott döntést hozhassunk a program komponenseinek szétosztásáról. A követelményeink meghatározzák az elfogadható késleltetés mértékét, illetve minél nagyobb sávszélességet szeretnénk a rendszer kihasználtságának maximalizálásához.

A mikrokontrolleren általában több olyan elem is található, ami befolyásolja az IPC teljesítményét. Ilyen például a gyorsítótárak hatása, amely esetén biztosítani kell a gyorsítótár-koherenciát. Sokszor több eltérő memória, más-más órajeltartományokban is elérhető, amik között lehetnek sebességbeli különbségek. A processzorok órajele és számos további tényező hatását is érdemes megvizsgálni az optimalizációhoz, valamint általánosan érvényes következtetéseket is levonhatunk az eredmények alapján.

Az STM32H745 kétmagos mikrokontroller egy heterogén architektúrájú eszköz, amelynél ezeket a szempontokat megvizsgáltam. Ennél a feladatokat egy nagyobb és egy kisebb teljesítményű mag között szeretnénk megosztani. A FreeRTOS beágyazott operációs rendszert alkalmaztam, mivel ez egy népszerű és bevált OS, ami támogatja az eszközt. A mikrokontroller mindkét magján külön operációs rendszer futott AMP szerint, ebben a felállásban szerettem volna a kommunikáció tulajdonságait megállapítani.

1.5. A dolgozat felépítése

A 2. fejezet áttekintést nyújt a felhasznált technológiákról, mint az operációs rendszer, az időmérés lehetőségei mikrokontrolleres környezetben, valamint a mikrokontroller releváns komponensei.

A 3. fejezet bemutatja a méréshez használt szoftver felépítését mind a mikrokontroller, mind a vezérlésre és az eredmények feldolgozására használt PC oldaláról.

A 4. fejezetben bemutatom a mérés eredményeit, a mért minták jellemzőit, az értelmezéshez felhasznált lineáris modellt, valamint a megvizsgált paraméterek hatását a kommunikációra.

Az 5. fejezetben összefoglalom az eredményeket és a továbbfejlesztési lehetőségeket.

2. fejezet

Kapcsolódó technológiák áttekintése

2.1. Operációs rendszerek összehasonlítása

Az asztali operációs rendszerek esetén általános célú, előre nem ismert alkalmazásokat viszonylag nagy teljesítményű hardveren futtatunk. A számítógép indításakor az operációs rendszert töltjük be, ami egységes interfészt biztosít az erőforrásokhoz. Az alkalmazásokat az operációs rendszer dinamikusan ütemezi, a platform erőforrásait is az OS (operating system, operációs rendszer) nyújtotta szolgáltatásokon keresztül lehet elérni. Alapvetően ezek az operációs rendszerek széles területen nyújtanak szolgáltatásokat, általános használatra vannak optimalizálva.

A beágyazott operációs rendszerek ezzel szemben az alkalmazással együtt kerülnek letöltésre az eszközre, a kernelt a programunk futásának megkezdése után indítjuk, ami ezután ütemezi a programunkat. Az OS olyan alkalmazásokra optimalizált, ahol az időzítésre hard vagy soft korlátokat garantálni kell, valósidejű ütemezésre van szükség, korlátozott mennyiségű erőforrás áll rendelkezésre, és specifikus feladatra készített programot futtatunk. A szoftverre általában szigorú követelmények vonatkoznak, az időzítés mellett biztonság és megbízhatóság szempontjából.

Asztali OS-ek esetén általában SMP-t alkalmazunk, az általában heterogén processzormagok között dinamikusan ütemezve a taszkokat. A taszkok közötti kommunikáció az operációs rendszeren keresztül, szabványos megoldásokkal lehetséges.

A beágyazott OS-ek esetén heterogén architektúrákon általában AMP-t előnyös alkalmazni, minden heterogén magon külön operációs rendszert, vagy bare metal kódot futtatva. A heterogén magokon itt is használható SMP. Az eltérő operációs rendszer példányok között is meg kell valósítani az IPC-t [8]. Ezek az operációs rendszerek a magok közötti kommunikációra minimális támogatást nyújtanak, ami azt jelenti, hogy biztosítanak IPC könyvtárat, amiben például bizonyos taszkok közötti kommunikációra használható objektumok módosíthatóak az OS-ben elérhető függvények segítségével, és használhatóak IPC-re. Azonban a kommunikációt ebben az esetben is nekünk kell megvalósítani általában megosztott memóriában, és minden ehhez kapcsolódó hardveres konfigurációt figyelembe kell vennünk. Kezelendő a gyorsítótár-koherencia, vagy processzorok közötti jelzések beállítása. Ezekre általában nem érhető el részletes útmutatás, természetesen az OS ezeket a részleteket nem rejti el. A heterogén magok közötti statikus feladatelosztás miatt az előre ismert erőforrásigényeket szabadon optimalizálhatjuk, az adott alkalmazásra a megfelelő elrendezést alakíthatjuk ki. Ehhez azonban meg kell vizsgálni a rendszer teljesítményét.

Összefoglalva az asztali és beágyazott operációs rendszerek teljesen eltérő programozási modellt használnak. Az asztali rendszerekben megismert tulajdonságokból nem von-

hatunk le egyértelműen következtetéseket a beágyazott operációs rendszerekben várható működésről [9]. Az asztali OS-ek nagyteljesítményű beágyazott rendszerekre szánt verziói is eltérő teljesítményt mutatnak az egyszerűsített megvalósítás miatt [10].

2.1.1. FreeRTOS

A FreeRTOS egy open-source (nyílt forráskódú) beágyazott operációs rendszer, amely a legnépszerűbbek közé tartozik [11, 12]. Jól dokumentált API-val (application programming interface, alkalmazás programozási felület) rendelkezik, könnyen alkalmazható és kis erőforrásigényű.

Az OS futásához csak három alapvető funkcionalitást biztosító, valamint egy platform specifikus port C fájlra van szükség. Minden további funkció használatához külön a projekthez adhatók a forrásfájlok. Az operációs rendszert az alkalmazásunkkal együtt kell lefordítani. Az operációs rendszert a kódban az ütemezőt elindító függvényhívással lehet futtatni.

A FreeRTOS számos kommunikációs és szinkronizációs objektumot tartalmaz, például flag-ek, mutexek, szemaforok, üzenetsorok, Stream Buffer és a Message Buffer. Az asszimetrikus többmagos rendszerben kommunikációhoz a Stream Buffer és Message Buffer használható. Ezek közül a Message Bufferrel végeztem vizsgálatokat, amiben egy útmutató nyújt segítséget [13]. Ennek segítségével változtatható méretű üzeneteket tudunk küldeni. Az üzenetek küldése megosztott memórián keresztül lehetséges, az üzenetek másolásával küldéskor és fogadáskor egyaránt. A sebesség és memóriaigény szempontjából igen kedvező no-copy (másolás nélküli) megvalósításhoz nem érhető el támogatás a Message Buffer használatával, valamint a másoláshoz nem tudunk DMA (direct memory access) vezérlőt sem használni.

2.2. Kommunikáció teljesítményének mérése mikrokontrolleren

Az IPC teljesítményének vizsgálata során a futásidőt és a memóriahasználatot lehet szükséges megvizsgálni. A kommunikációs objektum memóriahasználatát nincs módunk befolyásolni, egy kész implementációt szeretnénk felhasználni, amit az operációs rendszer biztosít, ezért ezt nem vizsgáltam meg a méréseim során. A memóriahasználatot fordítási időben is ismerhetjük, IPC esetén minden szükséges objektumot statikusan allokálunk megosztott memóriaterületen. A memóriahasználatot befolyásoló tényező az üzeneteket tároló buffer mérete, ami meghatározza a küldhető üzenetek méretét, ezért ez alapján érdemes megválasztani a követelményeinkkel összhangban.

A futásidő mérése azonban szükséges, amire több lehetséges módszer kínálkozik. Egy opció kimeneti jelszintek vizsgálata [14], például GPIO (general-purpose input/output) kimenetek logikai szintjének váltásával a mérni kívánt kód körül [15]. Ezzel az eljárással pontos órájú mérőeszközt tudunk a mérésre használni, viszont a mérés nehezen automatizálható és műszert igényel.

Egy másik megoldás profiling (profilozó) eszközök használata. Ilyenkor a kód instrumentálásával vagy mintavételezésével egy CPU-n rögzíthetjük metódusok futásának arányát, vagy pontos futásidejét is [16, 17]. Megfelelő támogatással lehetőség van így a hardver pontos teljesítményét is megmérni [18]. A magok közötti kommunikáció teljesítményének mérésére a profiling eszközök nem megfelelőek, továbbá a profiling eszközök és a processzorban elérhető támogatás mélyebb ismeretét igénylik.

A futásidő mérésére PC-s környezetben használt benchmarking (teljesítménymérő) könyvtárakhoz hasonlóan a rendszerben elérhető időzítők alapján is lehetséges [3, 17]. A futásidő egy időzítő értékének olvasásával történik, először a mérni kívánt kód kiindulási

időpontjában, majd a befejeztekor. A futásidő a két mért érték különbségeként számítható. Egy üres kódrészlet mérése esetén az elvárt futásidő nulla. Emiatt érdemes a futásidő értékének rögzítése üres kód mérése esetén, amellyel a későbbi mérések kompenzálhatók.

Ezzel a megoldással a mérések könnyen automatizálhatók, így nagy számú mérést gyorsan el lehet végezni, és a mérés további paraméterei is kézi beavatkozás nélkül, a mérésre használt script-ből változtathatóak. A mérési eredményeket érdemes nagy számú mérés elvégzésével rögzíteni, majd pedig statisztikai eszközökkel is kiértékelni, így az esetleges anomáliák gyorsabban felderíthetőek [19]. Mivel az optimalizáló fordítók a kódot módosíthatják, így meg kell győződni arról, hogy az időméréshez használt utasításokat a fordító nem távolította-e el a mérni kívánt kódból [17].

A mérések során a fordító által legnagyobb mértékben optimalizált kódot érdemes vizsgálni, mivel a valós környezetben futó alkalmazás esetén is ezt várhatjuk. A mérések kiértékelésekor a futásidő alapján a kommunikáció késleltetését vizsgáltam, ami az elérhető legjobb válaszidőt befolyásolja, valamint az adatátviteli sebességet, aminek a rendszer kihasználtságára lehet hatása.

2.3. Az STM32H745 mikrokontroller

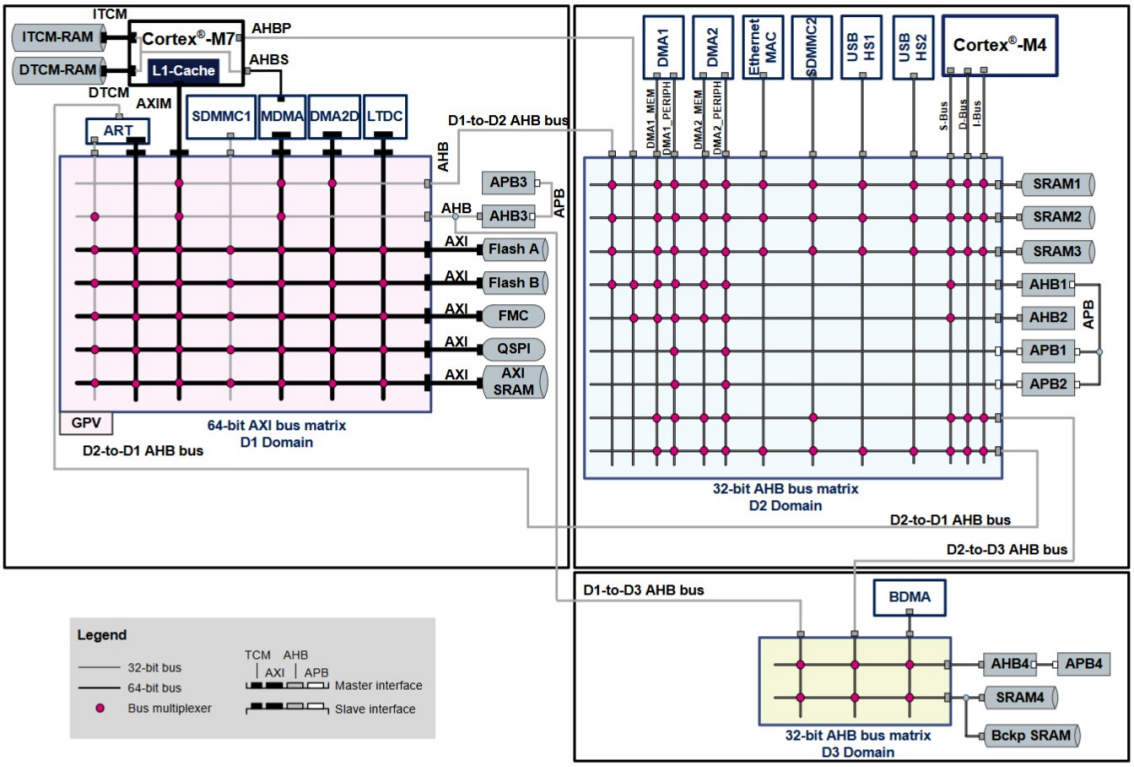
A méréseim során az STM32H745 mikrokontrollert használtam, ami egy heterogén architektúrájú kontroller, egy ARM Cortex-M7 és egy ARM Cortex-M4 magot tartalmaz [20][21]. Az IPC szempontjából érdemes megvizsgálni a rendszerben található memóriákat, az elérhető órajel-frekvenciákat, a processzorok közötti szinkronizáció és jelzés lehetőségét, valamint a perifériák kezelésének módját.

2.3.1. Memóriák elrendezése

A rendszert alkotó komponensek külön tápellátási doménben helyezkednek el, ezért külön-külön lekapcsolhatóak, ha szükséges, így optimalizálva a fogyasztást (2.1. ábra). A három domén elnevezése D1, D2 és D3. A M7-es mag a D1 doménben található, mellette nagy sebességet igénylő perifériák. A D2-ben az M4-es mag, kommunikációs perifériák és időzítők találhatóak. A D3-ban pedig kiegészítő funkciók helyezkednek el, például a reset és órajelek vezérlése. Minden doménben külön-külön buszmátrix található, valamint minden buszmátrix közötti kapcsolat is ki van építve. Minden buszmátrix legfeljebb 200 MHz frekvenciával képes működni.

A rendszerben található SRAM (static random-access memory) a három különböző tápellátási tartományban elosztva helyezkedik el. A D1 doménben az M7-es mag a 64 bites AXI (Advanced eXtensible Interface) buszon keresztül közvetlenül elérheti az itt található memóriát. A D2-es doménben elhelyezkedő buszvezérlők a két tartomány buszai között elhelyezkedő kapcsolaton (*D2-to-D1 bus*) keresztül érik el az itt található slave-eket, így a flash memóriákat is. A hozzáférés az ART (adaptive real-time) gyorsítón keresztül történik. Ez az AHB switch (Advanded High-performace Bus) mellett az utasítások hozzáférésének gyorsításához teljesen asszociatív cache-t is biztosít, összesen 64 256 bites sort. Az AXI mátrixba minden slave-hez beépítve megtalálhatóak az AXI és AHB közötti váltáshoz szükséges buszhíd funkciók. Az M7-es processzorban egy L1 szintű cache található, külön 16 kB adat (*DCACHE*) és 16 kB utasítás cache (*ICACHE*) is. Minden cache 32 bájtos sorokat tartalmaz, az adat 4 utas csoport-asszociatív (set-associative), az utasítás cache pedig 2 utas csoport-asszociatív.

A rendszerben található MPU (memory protection unit) segítségével memória-tartományok különböző attribútumokkal láthatóak el. A magok közötti kommunikáció szempontjából fontos, hogy a memóriára megadható, hogy gyorsítótárazható, illetve megosztható-e. Mivel a mikrokontroller nem támogatja a hardveres cachekoherenciát,



2.1. ábra. A rendszer busz architektúrája [21]

ezért ezt vagy a megosztott memóriaterület szoftveres gyorsítótár érvénytelenítésével és tisztításával érhetjük el, vagy az MPU-ban a terület gyorsítótárazásból való kizárásával. A rendszeren a támogatás hiánya miatt a megosztható memória ekvivalens a gyorsítótárazásból való kizárással.

A D2 tartományban elhelyezkedő memóriák AHB buszon keresztül érhetőek el, az M4-es magból közvetlenül, D1 doménből pedig egy buszhídon keresztül (*D1-to-D2 bus*). A D3 domén memóriája az ebben a tartományban található AHB buszon keresztül használható. A memóriák elérése várakozási ciklusok nélkül lehetséges.

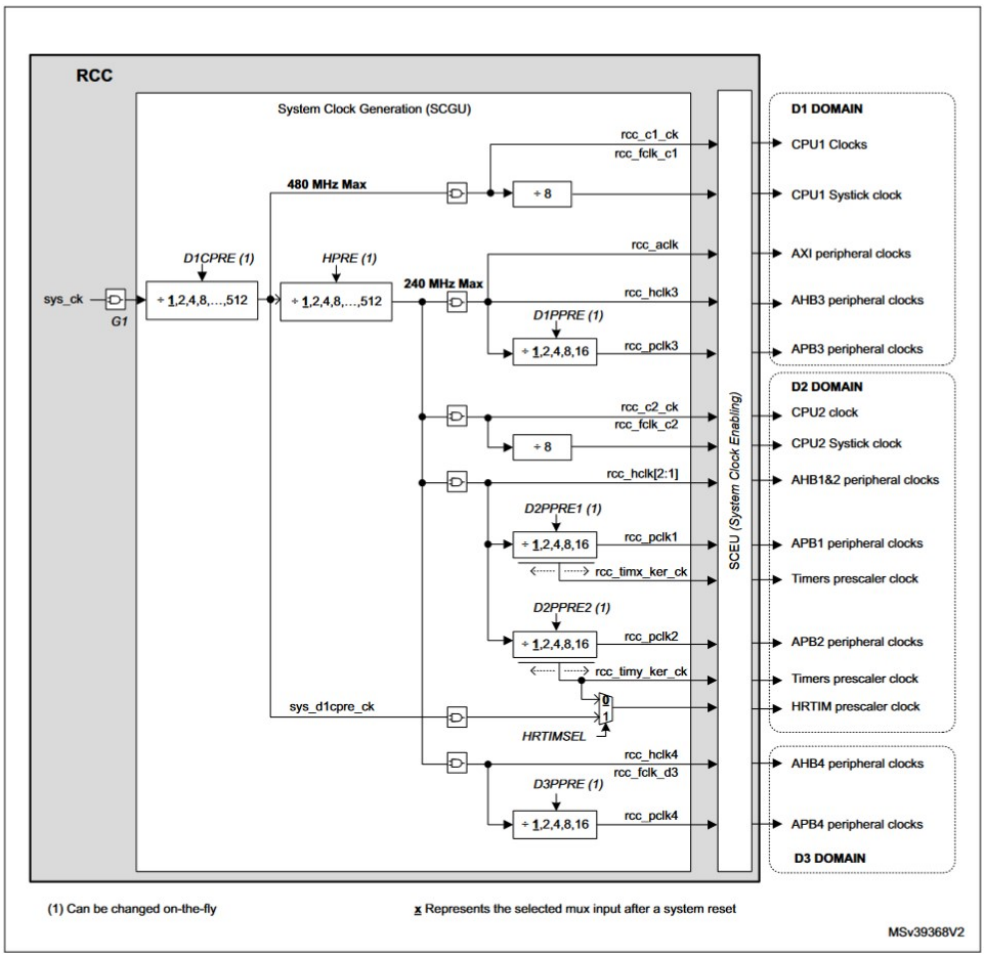
A rendszerben található nemfelejtő memória két független bank flash-ként került kialakításra. A flash memória 266 bites szavakként van szervezve, amiből 10 hibajavító bit.

2.3.2. Órajel-konfiguráció

A CPU-k (central processing unit), buszok és perifériák órajelének előállítása egy közös órajelből (*sys_ck*) történik órajelosztók segítségével (2.2. ábra). Az M7-es mag órajelének frekvenciája 480 MHz-ig, a többi komponensé, így a M4-es magé pedig 240 MHz-ig növelhető. Mivel az M7-es mag órajeléből állítjuk elő osztással a többi órajelet, így ennek megfelelően alakulhatnak a lehetséges kombinációk. A buszok órajelei megegyezhetnek a processzorok frekvenciájával, azonban legfeljebb 200 MHz-esek lehetnek. Az időzítő perifériák órajele az APB buszok frekvenciájánál lehet magasabb, a mérések során ez megegyezett az M4-es CPU frekvenciájával, valamint ezekkel egyeztek a buszok beállított frekvenciái is.

2.3.3. Processzorok közötti kommunikáció

Az IPC kommunikáció megosztott memórián keresztül lehetséges. Megosztott erőforrások védelme, valamint szinkronizáció hardveres szemaforral (HSEM, hardware semaphore) le-



2.2. ábra. A buszrendszer és processzor magok órajel-generálása [20]

hetséges. Ezeket atomi módon tudjuk lefoglalni. A magok közötti jelzésre továbbá használható szoftveresen generálható megszakítás. Ehhez az EXTI (extended interrupt and event controller) periféria használható. Ennek segítségével szabadon konfigurálható vonalakat szoftveresen bebillenthetünk, ami megszakítást generál, ha a megfelelő magot erre a jelzésre érzékenynek állítjuk be.

2.3.4. Periféria allokáció és hozzáférés

A perifériák nagy része külön busz és kernel órajellel rendelkezik, amik rendre a buszon való kommunikációra és a periféria funkciójának biztosítására szolgál. Ezzel az elrendezéssel lehetséges a magok és a busz mátrix órajel-frekvenciájának megváltoztatása a periféria befolyásolása nélkül. A perifériák a processzormagokhoz rendelhetőek, így az adott mag állapotát követik, tehát használható egy másik doménben található periféria a tartomány alacsony fogyasztású módba kapcsolásakor is. Ha egyszerre két processzorhoz rendeljük valamelyik perifériát, akkor az alkalmazás feladata az ütközés elkerülése.

3. fejezet

A mérési elrendezés összeállítása

A kommunikáció tulajdonságainak mérése alatt mindkét magon FreeRTOS futott, különkülön ütemezéssel.

A mérési elrendezés felépítéséhez a legfontosabb előkészíteni a mérni kívánt kommunikációs objektumot, ami a Message Buffer volt. Megvalósítandó továbbá a futásidő méréséhez szükséges környezet. A futásidő vizsgálatához az operációs rendszerben taszkokat kellett létrehozni, amikben az operációs rendszer által nyújtott szolgáltatásokat is lehetőségem volt felhasználni. A projektben a mérést elvégző taszkokon kívül nem hajtódik végre más feladat, így a kommunikáció tulajdonságait optimális esetben vizsgálhatjuk meg.

A PC oldalán a mérés vezérlését és az adatok feldolgozását Python scriptek végzik.

A méréshez használt mikrokontrolleren futó projekt kódja és az adatokat megjelenítő és feldolgozó scriptek elérhetőek egy-egy Github repository-ban [22, 23].

3.1. Futásidő mérésének előkészítése

A futásidő vizsgálatához a mikrokontrolleren egy hardveres időzítő periféria számlálóját olvasom mindkét magról. Az időzítő perifériát az egyik mag konfigurálja. Az idő pontos méréséhez mindkét mag ennek a perifériának a számlálóját olvassa, a kettő különbségéből számolva az eltelt időt. Ehhez a másik magon is szükség van legalább a perifériát azonosító struktúra feltöltésére. Ezt a generált kód nem tette meg annak ellenére, hogy beállítható, hogy mindkét domén a perifériához hozzáfér, és ezek közül az egyik inicializálja. Az inicializációt nem végző magon a `TIM_HandleTypeDef` típusú, időzítőt azonosító struktúra `Instance` mezőjét beállítva a periféria kezdőcíme a memóriában már megfelelő, és így a számláló értéke is helyesen került olvasásra.

3.2. Az IPC kommunikáció megvalósítása

A többmagos kommunikációhoz a Message Buffer-hez érhető el támogatás, ezért ezt részletesebben bemutatom [13]. Ennek használatához össze kell állítani egy szoftverprojektet, amelyben az operációs rendszer elérhető. Ezután az operációs rendszer dokumentációja alapján, ami részletes útmutatást nyújt, megvalósítható az IPC kommunikáció.

3.2.1. Szoftverprojekt előkészítése

Az operációs rendszer dokumentációjában megtalálható egy példaprojekt, amely bemutatja a Message Buffer kommunikációs objektum használatát az STM32H745 mikrokont-

rolleren [11]. Lehetséges ennek a projektnek a felhasználása kiindulási pontként, azonban így a perifériák konfigurációját kézzel, automatizálás nélkül kell elvégezni.

A konfigurációt megkönnyíti a hardverhez elérhető STM32CubeMX szoftver, aminek segítségével az operációs rendszer a projekthez adható, valamint a hardver perifériák egyszerűen beállíthatóak. Az STM32CubeMX az STMCubeIDE fejlesztői környezetbe integrálva is elérhető.

3.2.2. FreeRTOS Message Buffer használata IPC kommunikációra

A FreeRTOS operációs rendszer dokumentációja a magok közötti kommunikációhoz a Message Buffer vagy Stream Buffer objektumok használatát javasolja.

A Stream Buffer kommunikációs objektum a FreeRTOS-ban egy író és egy olvasó taszkra optimalizált. A segítségükkel egy bájtfolyam küldése lehetséges [11, 13]. Erre az implementációra épül a Message Buffer objektum, amelyben már diszkrét, változtatható méretű üzeneteket tudunk kezelni. Egy elküldött üzenet csak ugyanakkora méretű üzenetként fogadható.

3.2.2.1. Üzenetek küldése a magok között

Az útmutató a Stream Buffer és Message Buffer objektumok az egymagos viselkedésének megváltoztatásához az `sbSEND_COMPLETED()` makrót felüldefiniálja. Egymagos esetben ennek hívása az ugyanabban az operációs rendszerben várakozó taszkokat értesíti.

A makró feladata többmagos környezetben a többi mag értesítése, ami megtehető szoftveres megszakítás generálásával a processzorok között. Ennek hatására a fogadó magon vizsgálandó, hogy várakozott-e taszk üzenetre, és ha igen, akkor szükség van-e ütemezésre.

Ahhoz, hogy párhuzamosan több Message Buffert rugalmas módon tudjunk IPC-re használni, érdemes az `sbSEND_COMPLETED()` makróban először egy további megosztott vezérlő Message Bufferbe írni a megváltozott buffer címét. Ha a címet rögzítettük, szoftveres megszakítást kell végrehajtani. Az interrupt hatására a fogadó oldalon ki kell olvasni az üzenetet tároló Message Buffer címét, majd az erre várakozó taszkokat ütemezni ha szükséges.

3.2.2.2. IPC Message Bufferek allokációja

A magok közötti kommunikációra a mikrokontrollerekben általában, így az STM32H745 esetén is megosztott memóriát tudunk használni, ezért ebbe egy mindkét doménből elérhető és ismert memóriacímre kell minden IPC objektumot elhelyezni.

A Message Bufferek közül az üzenetküldésre és a vezérlésre használt objektumot egyaránt meg kell osztani a domének között. FreeRTOS-ban létezik dinamikus allokáció, ami erre a célra nem megfelelő, ezért fordítási időben, statikus allokációval kell a megosztott Message Buffer-nek helyet foglalni. Ilyenkor az üzenetek tárolására használt tömböket is természetesen megosztott memóriában kell elhelyezni.

Az időmérés kezdetekor és végeztével rögzített hardveres számláló értéket külön magokon rögzítjük, ezért ezeket szintén egy egyszerű megosztott változóban adtam át a két fél között.

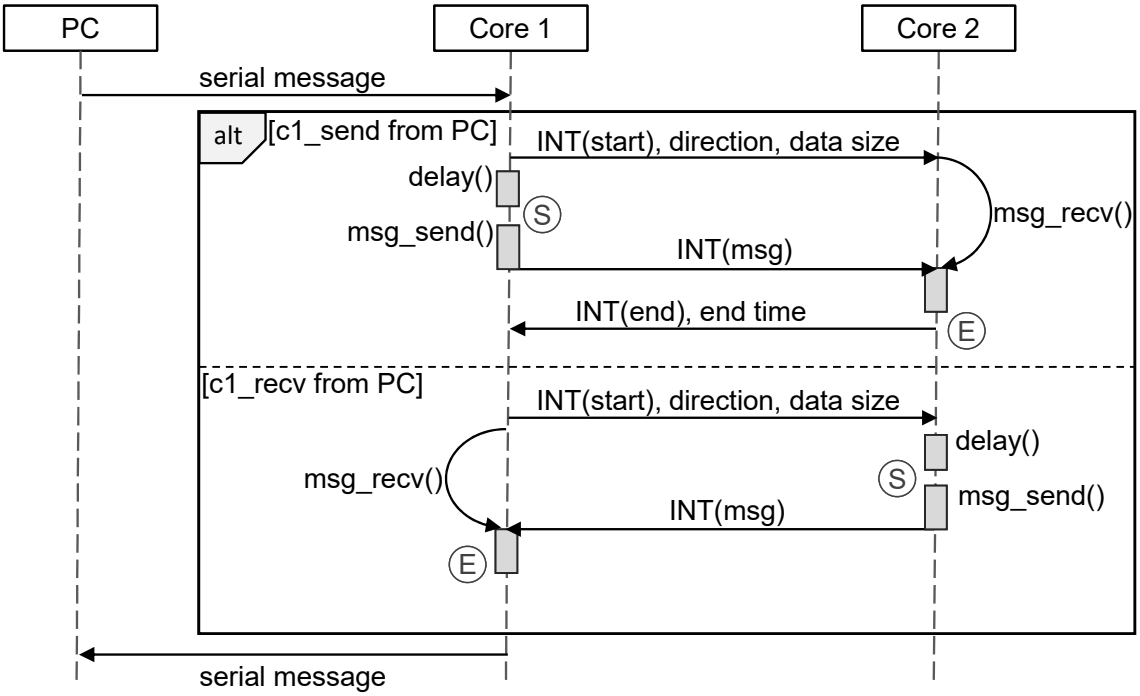
A megosztott változók helyét a memóriában tipikusan egy linker script segítségével, továbbá a kódban fordító specifikus direktívákkal lehet befolyásolni. Mindenképp szükséges, hogy a megosztott memóriát felhasználó processzorok ugyanahhoz az objektumhoz férjenek hozzá, így kézenfekvő előre ismert címre elhelyezni őket. A mérések során erre a rendszerben elérhető memóriák összehasonlításához is szükség volt, a cím megadásával megválaszthatjuk a használt memóriát.

Az STM32H745 mikrokontroller esetén a memória címkiosztása mindkét mag felől megegyezik, ezért azonos memóriacímet kell használniuk a megosztott változók olvasásakor és írásakor. Ehhez a GCC fordító esetén egy attribútummal megadható a memória szegmens. Ezt a linker scriptben a memóriában kívánt címre elhelyezhetjük, valamint biztosítani kell, hogy a szegmensen belül az objektumok ugyanolyan sorrendben kerüljenek elhelyezésre mindkét magra fordítandó program linkelésekor. Ügyelni kell arra, hogy a szegmensünk ne kerüljön más automatikusan elhelyezett tartalommal megegyező címre, ez nehezen felderíthető hibát okozhat, ha a build során nem értesülünk róla hibaüzenet formájában.

3.3. Mérési elrendezés a mikrokontrolleren

Az mikrovezérlőn a mérést és a soros kommunikációt a PC-vel is az egyik mag vezérli. A soros kommunikációra használt periféria kezelését ebben az esetben csak az egyik mag végzi, az időzítővel szemben itt nem szükséges minden mag hozzáférése. Ez általában előnyösebb megoldás, nem merülnek fel kölcsönös kizárással kapcsolatos problémák, a konzisztens küldött adatot egyszerűbb biztosítani, mint megosztott perifériák esetén.

A mérés kezdő és végső időpontja alapján az időtartam kiszámítható, amelyet ezután soros porton lehet továbbítani a PC felé.



3.1. ábra. A mikrokontrolleren futó program szekvenciadiagramja pszeudokóddal

3.3.1. Általános felépítés

A szekvenciadiagramról (3.1. ábra) leolvasható a mérés felépítése. A PC-ről soros porton való vezérlés során (`serial message`) beállíthatók a mérés paraméterei, és a mérés elindítható. Az eredmények továbbítása is így lehetséges. Az `S` és `E` karakterek az időmérés kezdetét és végét jelzik az ábrán.

A kommunikációtól irányától függően két alternatíva lehetséges. A mikrokontrolleren megvalósított programban egy minta mérése a következőképpen történhet:

Ha a mérést vezérlő mag küldi az üzenetet (`c1_send`), akkor erről értesíteni kell a másik magot (`INT(start)`), amely ekkor az üzenet fogadását megvalósító hívással az üzenet érkezésére várakozik (`msg_recv()`). Ebben az esetben a vezérlő mag rövid ideig várakozik, hogy a fogadó oldalon mindenképp elegendő idő legyen a fogadás megkezdésére. A vezérlő mag ezután az időméréshez rögzíti a közös óra számlálóját, majd elküldi az üzenetet (`msg_send()`). A fogadó mag szintén rögzíti az óra állapotát az üzenet fogadása után.

Ha a mérést vezérlő mag az üzeneteket fogadja (`c1_recv`), akkor az elrendezés nagyon hasonló. Az üzenetet küldő magot értesíteni kell, amely ezután rövid ideig blokkol, hogy az üzenetet fogadó hívás megtörténhessen. Ezután rögzíti a kezdés időpontját, majd elküldi az üzenetet. Az üzenet fogadását követően a soros kommunikációt fogadó mag rögzíti az aktuális időpontot.

Amikor a mérés kezdetét a vezérlő mag jelzi, akkor továbbítani kell a mérni kívánt üzenet méretet és a kommunikáció irányát is (`direction` és `data size`), hogy a vezérelt mag megfelelő kódot hajthasson végre. A mérés végeztével a vezérelt magon rögzített időpont megosztott változóban rendelkezésre áll, a futásidő ez alapján számítható.

Mivel az ismertetett mérési elrendezéssel egyetlen küldés idejét rögzítjük, így a mérési adatok eloszlását is megvizsgálhatjuk. Az mérési adatok feldolgozása során kell az adatokat átlagolni, valamint az eloszlás további tulajdonságait is lehetőségünk van kiértékelni.

3.3.2. Megvalósítás az STM32H745 mikrovezérlőn

A mérés vezérlése a PC oldaláról virtuális soros porton keresztül történik, ami a mikrokontrolleren egy USB-UART hídon keresztül UART kommunikációként valósul meg. A fogadott üzeneteket egy állapotgép dolgozza fel. A mérés konfigurálásához megadható paraméterek a mérés ismétlésének száma, a kommunikáció iránya, valamint a megmérni kívánt üzenet mérete. A kommunikáció iránya `'s'` és `'r'` karakterekkel, a méretek pedig megfelelő sorrendben decimális számokkal és egy lezáró `\r` karakterrel jelezhetőek.

A magok között Message Buffer segítségével küldött üzeneteket kisebb hibaellenőrzésre alkalmas tartalommal kell feltölteni. A buffer legelején stringként az üzenet méretét helyeztem el, majd egyforma adattal, ciklusonként növekvő bájt értékekkel van kitöltve az üzenet fennmaradó része. A hibaellenőrzés ezekre az üzenetekre a méret kiolvasása, aminek meg kell egyeznie a fogadott bájtok számával, valamint az üzenet utolsó bájtjának ellenőrzése.

A késleltetés alkalmazása az üzenet küldése előtt biztosítja, hogy a fogadó oldalon az üzenetet fogadó hívásban blokkol a fogadó taszk. A késleltetés hosszának elegendőnek kell lennie ahhoz, hogy a konfigurált órajelpár mellett a fogadónak elegendő ideje legyen a függvényhívásra, enélkül a mért futásidőkben egy kisebb mértékű állandó hiba jelenne meg, mivel a küldő mag hamarabb elküldheti szerencsétlen esetben az üzenetet, mint ahogy a fogadást megkezdenénk. A programomban erre az elérhető legkisebb, 1 ms-os késleltetés természetesen elegendő volt. Mivel ez az órajelekhez képest hosszú időtartam, így a méréshez szükséges idő megnövekszik.

A magok közötti jelzésre a mérés indításakor és befejeztekor a magok közötti szoftveresen kiváltható megszakításokat használtam. Ez egy könnyen felhasználható, általában rendelkezésre álló megoldás. A megszakítás kezelő rutinban az operációs rendszerben elérhető szinkronizációs megoldásokkal lehet értesíteni a várakozó taszkot. A mérés vezérlésére a magok között használt változók, mint a mérés iránya, mért üzenet mérete, valamint az időmérés kezdő és végső időpontja, megosztott memórián keresztül kerülnek átadásra.

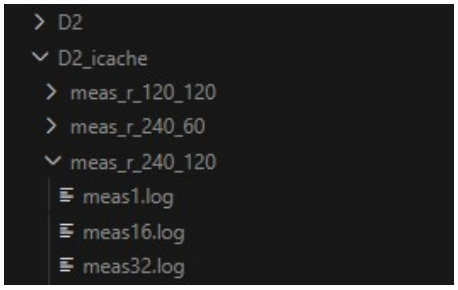
Az időmérés során a számlálók olvasása miatt jelentkező ofszetet megmértem, elhanyagolható, pár órajelnyi értékek jelentkeztek.

3.4. Mérési elrendezés a PC-n

A PC-n a Python nyelven megírt scriptek feladata a mérés vezérlése, majd virtuális soros porton keresztül a fogadott mérési eredmények mentése. A rögzített adatok segítségével a feldolgozás során az eredményeknek kiértékelhetőnek kell lennie. A Python nyelv megfelelő választás, mivel az ábrázolás, az adatok feldolgozása és a soros kommunikáció mind nagyon gyorsan megvalósítható a nagy számú elérhető modulnak köszönhetően.

3.4.1. Soros kommunikáció és a mérési eredmények rögzítése

A mikrovezérlőn a könnyű kezelhetőség érdekében a vezérlés során minden fogadott adatot visszaküldünk. A PC-n minden fogadott adatot fájlba írva minden információ rendelkezésre áll a későbbi feldolgozáshoz. A méréseket tároló fájlokat hierarchikusan mappákba szervezve tároltam el. Minden adatméret esetén 1024 mérést végeztem, ezért ezt nem szükséges megkülönböztetni az eredmények rögzítésekor.



3.2. ábra. A méréseket tároló fájlok hierarchiája

Minden memóriakonfiguráció mérései egy mappában helyezkednek el. Ezen belül órajelpáronként és kommunikációs irányonként további mappák találhatóak. Egyetlen mérés egy fájlban található, és egy adott adatmérethez tartozik, ezt a fájl elnevezése is tükrözi.

3.5. Mérési eredmények feldolgozása

A fájlokban az időmérésre használt időzítő periféria által mért órajelperiódusok száma van rögzítve, valamint az időzítő órajel-frekvenciája. A későbbiekben a kommunikáció adatátviteli sebességét (sávszélesség, data rate) és késleltetését (latency) vizsgáltam. A számításuk a 3.1 és 3.2 egyenletekkel lehetséges.

$$latency = N_{clk} * \frac{1}{f_{clk}} \tag{3.1}$$

$$datarate = D * \frac{f_{clk}}{N_{clk}} \tag{3.2}$$

Ahol D az átvitt adat mérete például bájtokban, f_{clk} a méréshez használt számláló órajele, N_{clk} pedig a számláló alapján számított futásidő órajel periódusokban.

A mérési adatokra paramétereiben lineáris modellt illesztettem. A modell paramétereinek meghatározásához fel kell építeni a mérési adatok alapján a megfigyelések, valamint az ismeretlen paraméterek vektorát, amely optimális megoldását Least-Squares (legkisebb négyzetes hiba) értelemben keressük. A paraméterek az együtthatók mátrixának pszeudoinverz számításával meghatározhatóak, vagy kész implementációt is használhatunk. Ezek közül az utóbbi megoldással éltem, a modell meghatározását az 4.3. szakaszban részletezem.

4. fejezet

Kommunikáció teljesítménymérésének eredményei

A méréseket egy Nucleo-H745ZI-Q kártyán végeztem el, amin egy STM32H745 kétmagos mikrokontroller található. A szoftverek fordításához a CubeIDE fejlesztői környezet 1.11.2-es verzióját használtam. Mindkét magon 10.3.1-es verziójú FreeRTOS operációs rendszer futott.

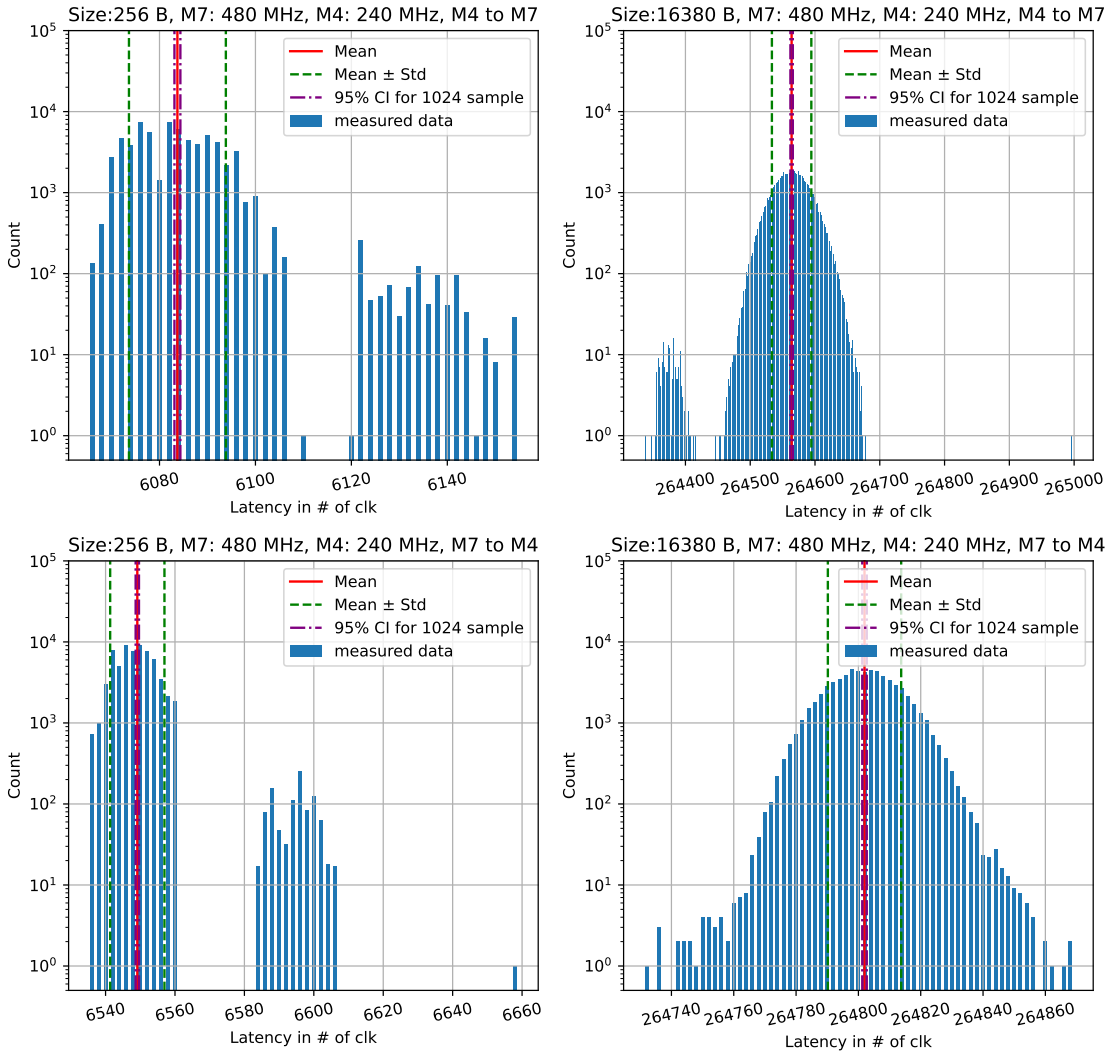
A Message Bufferek használatakor a magok közötti megszakítások a 0-ás és 4-es EXTI vonalon történtek. A mérés indításakor és végeztével jelzésre a 2-es és 3-as EXTI vonal megszakítását használtam. Az üzenetek tárolására foglalt buffer mérete 16384 bájt méretű volt, amiben legfeljebb 16376 bájt méretű adatot lehetséges küldeni. Időmérésre a TIM5 általános célú 32 bites időzítő perifériát használtam. Ennek a frekvenciája minden mérés során megegyezett az M4-es mag órajelének frekvenciájával. A processzorok órajelének változtatásakor 480 MHz frekvenciájú *sys_ck*-ból a *D1CPRE* és *HPRE* osztók változtatásával állítottam elő a kívánt frekvenciát (2.2. ábra).

A mérések során az optimális, best-case eredményeket mértem meg, mivel a projektben más feladat, nagy mennyiségű erőforrást igénylő taszk nem futott. Az ütemezés során nem okozhatott nagy eltérést az, hogy másik futásra kész taszkot lett volna szükséges az operációs rendszernek futtatnia. Ebből következik, hogy a késleltetésekre a mérési eredmények alsó, az adatátviteli sebességre pedig felső korlátot adnak.

4.1. Mérési minták vizsgálata

A mérés ismétlésszámának meghatározásához egy kiindulási mérést végeztem. Nagy mintaszámú (65536) mérés eredményei alapján hisztogramon ábrázoltam a minták eloszlását (4.1. ábra). Az ábrákon megjelenítettem a minták átlagát és az átlag körül egy szórás szélességű sávot. A végső, a későbbi futásidőmérések során alkalmazott mintaszám megválasztását befolyásolja a végeredmény pontossága és a méréshez szükséges időtartam. A pontosságot jellemezhetjük az átlag körüli konfidencia intervallummal. Az ábrán a 95 %-os konfidencia intervallum látható normális eloszlás alapján, 1024 ismétlésre. Minden további mérés során ezt az ismétlésszámot alkalmaztam. Ilyenkor a mérések elfogadható időt vettek igénybe, valamint a mért minták átlaga pontos eredményt ad. A konfidenciaintervallum nagyon keskeny, az 1024 méréssel kapható átlag értéke nagyon pontos lesz, ezért a későbbiekben minden ábrán a mérés átlagát ábrázoltam, valamint a minimum és maximum értékek közötti sávot. A konfidencia intervallum ennél minden esetben jóval szűkebb, az ábrákon nem lenne jól megjeleníthető.

A hisztogramon az első oszlopban, kis méretű üzenetek küldése esetén a mért értékek két sávban helyezkednek el. Ez magyarázható azzal, hogy valamilyen nagyobb késleltetést



4.1. ábra. A mért érékek eloszlása, az átlag és körülötte a szórás sávja, valamint 95 %-os konfidencia intervallum, ha 1024 értéket mérünk

okozó esemény esetén a küldés ideje megnövekszik, ez viszont nagyságrendekkel ritkábban fordul elő, mint a kisebb késleltetésű eset.

Nagy üzenetek hosszabb időtartam alatt történű küldése alatt (második oszlop) a mért értékek ingadozása szélesebb tartományon helyezkedik el. A csúcsok látszólag a normális eloszlás haranggörbéjét mutatják. A felső sorban, az M4-es processzorról az M7-esre üzenetet küldve itt is előfordulnak kis gyakorisággal minták, amik elkülönülnek a leggyakoribb esetek csúcsától.

4.2. A kommunikációt befolyásoló paraméterek

A vizsgálatok alatt igyekeztem miden olyan jelentős tényezőt figyelembe venni, ami a kommunikáció tulajdonságait az optimális esetet, terheletlen processzoron futó IPC-t befolyásolhatja. A mérések során ezek a paraméterek konfigurálhatóak.

A Message Buffer objektumok küldése során a küldő oldalon az üzenetet megosztott memóriába másoljuk, majd fogadáskor onnan ismét másolással fogadjuk. A másolás időtar-

tama, mindkét oldalon az adat méretével arányos. A teljes késleltetésben a két időtartam összege fog várhatóan megjelenni.

A processzorok órajel-frekvenciájával arányosan a teljesítményük természetesen növekszik. Az elérhető frekvenciák tartományában a memória elvileg várakozási ciklusok nélkül elérhető, az utasítások végrehajtása pedig előre ismert számú órajelciklust vesz igénybe. Ez alapján az elvárásunk az, hogy a kommunikáció késleltetése mindkét magon fordítottan arányos a processzor órajelének frekvenciájával.

A rendszerben több tápellátási doménben található RAM memória. Ezeket eltérő típusú és szélességű buszokon keresztül érhetjük el. A dokumentáció alapján a rendszer beható vizsgálata nélkül nehéz lehet megállapítani a memóriák közötti lehetséges sebességkülönbséget.

A kommunikáció iránya a fogadás és küldés műveletét cseréli fel a processzorok között, ha valamelyik több művelettel jár, akkor ezt a gyorsabb processzoron futtatva kisebb késleltetést várhatunk.

A Cortex-M7-es processzorban található L1 szintű gyorsítótár várhatóan felgyorsítja a futás, a kód időbeli és címtérbeli lokalitását kihasználva. A cache használata azonban nehezen felderíthető hibákat okozhat és alaposan megtervezett kódot igényelhet a koherencia biztosításának érdekében. A megosztott memóriát gyorsítótárazva rossz értékeket fogunk olvasni. A rendszerben a megosztott memóriaterületet az MPU segítségével kizártam a gyorsítótárazásból. Ehhez a *Cacheable* és *Bufferable* bitet 0-ra állítottam, a *Shareable* bitet pedig 1-re. [24]. Az *ICACHE* hatása nehezen becsülhető, a *DCACHE* pedig a megosztott memóriában történő műveleteket nem tudja gyorsítani, a további szükséges műveleteket viszont igen.

4.3. Lineáris modellillesztés

A rendszer pontosabb megismeréséhez a mért adatokra paramétereiben lineáris modellt illesztettem. Erre azért volt szükség, mert a mérési adatok számos paramétertől függenek, az összefüggések jobb megértése lehetséges, ha az eltérő konfigurációk esetén illesztett modellek optimális paramétereit is összevethetjük. Az esetemben a mérési eredmények vizsgálata során felmerült, hogy érdemes az eredményeket memóriakonfigurációnként és kommunikációs irányonként szétválasztani. Ezekre az esetekre az órajelektől és az adatmérettől függő modellt illesztettem.

A modell szerint a késleltetés ugyanabból a két tagból épül fel a mindkét processzor oldalán, a teljes késleltetés a fogadáshoz és küldéshez szükséges összege. A két tag egyrészt szükséges fix számú utasításból, ezen kívül pedig az adatok másolásából áll. Mindkét tag arányos az órajel reciprokával, a másolás pedig az adatmérettel is.

$$latency = b_{M7} * \frac{1}{f_{M7clk}} + d_{M7} * \frac{datasize}{f_{M7clk}} + d_{M4} * \frac{datasize}{f_{M4clk}} + b_{M4} * \frac{1}{f_{M4clk}} \tag{4.1}$$

A modellt a 4.1. egyenlet írja le, ahol b_{M7} és b_{M4} a modell paraméterei, amik az adatmérettől függetlenek, állandó összetevők tagjai, ezek az IPC során a mindenképpen végrehajtandó utasítások számával arányosak. A d_{M7} és d_{M4} az adatmérettel arányos, változó komponens paraméterei, az adat másolásához szükséges utasításoktól függenek. Az f_{M4clk} és f_{M7clk} a két processzor frekvenciája, *datasize* pedig a küldött adat mérete.

A kommunikáció késleltetésében a későbbiekben is állandó és változó tagként hivatkozok a két összetevőre. A modell az ábrázolás során felhasználható, a segítségével kiegészíthetőek az ábrán a mérési pontok becsült értékekkel.

A modellillesztés során az $\mathbf{y} = \mathbf{Kx}$ egyenlet optimális megoldását keressük legkisebb négyzetes hiba értelemben, ahol ebben az esetben az együtthatók a 4.2. egyenlet szerint alakulnak. Ahol a $latency_i$ ételek a mérés során mért késleltetés minták átlaga, a többi paraméter pedig a korábbiak szerinti értékek, amik a mérési ponthoz tartoznak.

$$y = \begin{pmatrix} latency_1 \\ latency_2 \\ \vdots \\ latency_n \end{pmatrix}, K = \begin{bmatrix} \frac{1}{f_{M7clk1}} & \frac{datasize_1}{f_{M7clk1}} & \frac{datasize_1}{f_{M4clk1}} & \frac{1}{f_{M4clk1}} \\ \frac{1}{f_{M7clk2}} & \frac{datasize_2}{f_{M7clk2}} & \frac{datasize_2}{f_{M4clk2}} & \frac{1}{f_{M4clk2}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{f_{M7clkn}} & \frac{datasize_n}{f_{M7clkn}} & \frac{datasize_n}{f_{M4clkn}} & \frac{1}{f_{M4clkn}} \end{bmatrix}, x = \begin{pmatrix} b_{M7} \\ d_{M7} \\ d_{M4} \\ b_{M4} \end{pmatrix} \tag{4.2}$$

A modellillesztést elvégezve a következő értékeket kaptam (4.1. táblázat). Minden paraméter növelésével a késleltetés növekszik, ezért a kisebb értékek jelentenek gyorsabb kommunikációt az állandó és változó tagban is. Az egyenletek felírásakor minden adatmérethez tartozó mérési pontot felhasználtam és 60 MHz feletti órajeleket, mivel a kezdeti vizsgálatoktól eltekintve minden mérést ezek a feltételek mellett végeztem.

Memória konfiguráció	$b_{M7}[1]$	$d_{M7}[\frac{1}{\mathrm{B}}]$	$d_{M4}[\frac{1}{\mathrm{B}}]$	$b_{M4}[1]$
M4-M7				
D1	2070.27	0.91	21.94	3925.35
D1 gyorsítótárral	2013.27	4.37	19.93	2156.89
D2	1903.32	4.91	12.78	3780.24
D2 gyorsítótárral	1982.52	4.74	13.58	1931.01
D3	1857.96	3.79	13.28	3888.44
D3 gyorsítótárral	1991.75	4.38	13.84	2017.16
M7-M4				
D1	1486.57	1.75	17.24	4065.78
D1 gyorsítótárral	1182.42	0.02	17.17	3208.31
D2	1896.73	2.22	12.22	3597.26
D2 gyorsítótárral	1286.97	2.86	8.18	2996.62
D3	1551.96	3.30	12.59	3921.64
D3 gyorsítótárral	967.56	0.03	16.10	3227.30

4.1. táblázat. A lineáris modell paraméterei.

Minden adatmérethez tartozó mérés esetén sok olyan pontra illesztünk, amikben a késleltetés változó összetevője nagy, ezért megvizsgáltam, hogy mennyivel kapunk az állandó komponensre jobb becslést kisebb méretekre való illesztéssel. Az átlagos négyzetes hiba értékek a modell által becsült és a mérési eredmények között legfeljebb 256 B adatméretek esetén az 4.2 táblázatban láthatóak. Az MSE_{256} a 256 bájtig illesztett modellhez, az MSE_{16376} pedig a minden adatra illesztett modell eredménye.

Az átlagos négyzetes hibaértékek nem térnek el jelentősen, ezért az összes adatra illesztett modellt használtam fel az ábrázoláshoz és értelmezésre is a 4.1. táblázat szerinti együtthatókkal.

4.4. A mérések eredményei

4.4.1. A fordító által végzett optimalizáció és az adatméret hatása

Az üzenet méretének hatását a kommunikáció teljesítményére a 4.2. ábra mutatja be. A mérési pontok mellett szaggatott vonallal látszik a sebesség becslése a modell szerint. A modell alapján nagyságrendileg helyes, durva becslést kapunk a viselkedésre.

Memória konfiguráció	MSE_{256}	MSE_{16376}
M4-M7		
D1	5.314	5.733
D1 gyorsítótárral	0.268	0.705
D2	3.726	3.741
D2 gyorsítótárral	0.372	0.830
D3	1.903	1.919
D3 gyorsítótárral	0.209	0.399
M7-M4		
D1	4.445	4.957
D1 gyorsítótárral	1.080	1.567
D2	3.404	8.431
D2 gyorsítótárral	0.719	0.910
D3	1.931	3.224
D3 gyorsítótárral	1.286	1.305

4.2. táblázat. Átlagos négyzetes hiba értékek legfeljebb 256 B méretű üzenetekre.

Az első sorban a késleltetésről (*latency*) látszik, hogy az üzenet méretével arányosan lineárisan növekszik. Kevés adat esetén (első oszlop) a késleltetésben állandó összetevő jelentős, nagy üzenetek esetén pedig az adatok méretével arányos összetevő. A változó tag a kommunikáció kódjában a másolás, az állandó pedig minden további kód lehet. Az adatátviteli sebességben (*datarate*) kis méretek esetén még az állandó összetevő meghatározó, a méret növelésével az sebesség folyamatosan növekszik. Nagy méretek esetén az elérhető legnagyobb sávszélességet is megfigyelhetjük.

Az optimalizált fordítás esetén a késleltetés állandó összetevője változik jelentősen, hasonlóan a gyorsítótár hatásához. A másolás sebességét ezzel nem tudjuk jelentősen befolyásolni, ezért a késleltetés meredeksége nem változik. Az optimalizált kódhoz tartozó mérési pontok hasonló görbét rajzolnak ki, mint debug kód esete, illetve közel illeszkednek a becsült lineáris esetre is.

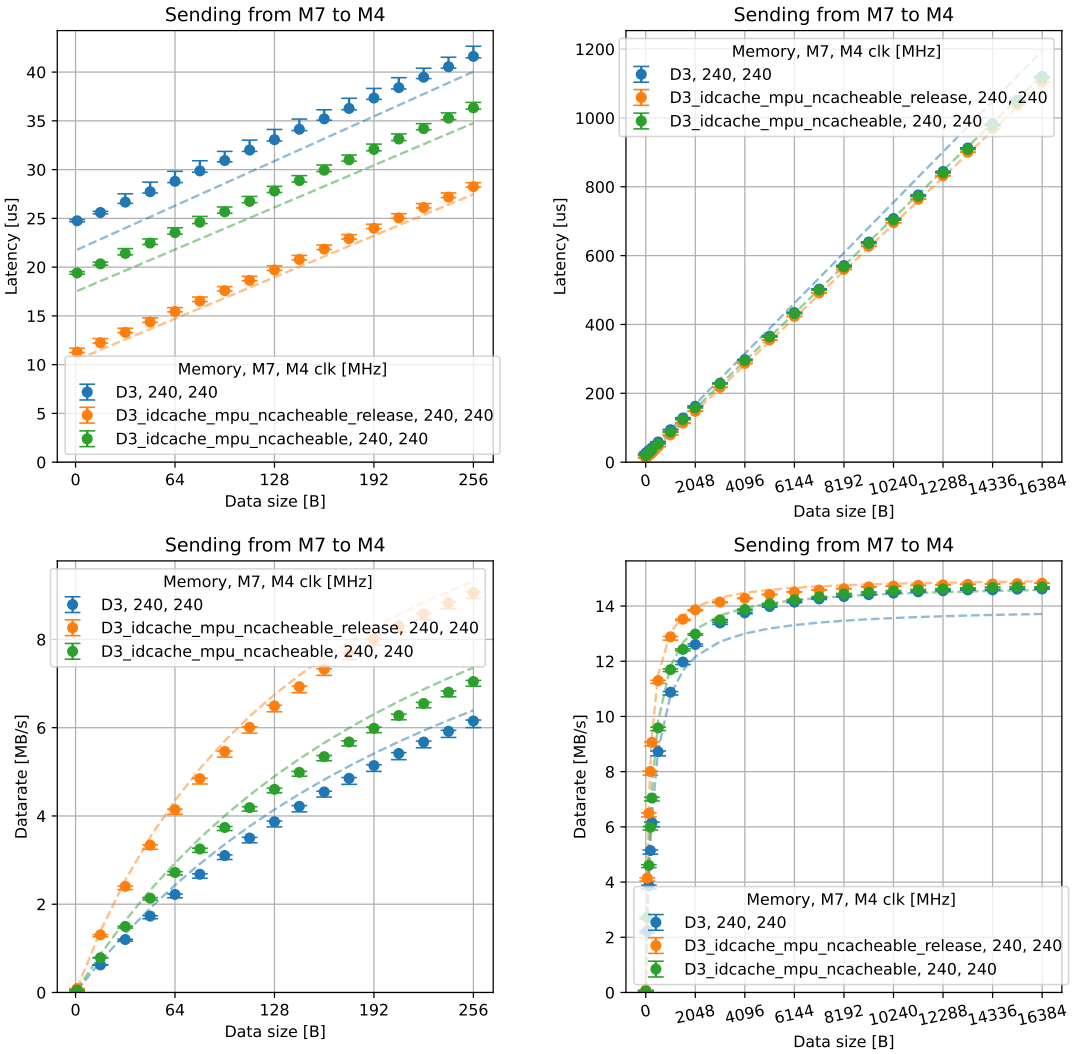
A továbbiakban a késleltetést kis adatméretek esetén ábrázolom, ezen az állandó összetevőt figyelhetjük meg, nagy méretek esetén pedig az adatátviteli sebesség szerint az elérhető leggyorsabb átvitelt tudjuk megvizsgálni.

A gyakorlati alkalmazásban várható, optimális teljesítmény megállapításához érdemes a leginkább optimalizált kódra elvégezni a méréseket. A következő mérések azonban minden további ábra esetén optimalizálatlan, debug kódot mutatnak, amikből a ábra alapján a kommunikáció paraméterfüggésének jellegét helyesen állapíthatjuk meg. A D3-ban található memória görbéi optimalizált kód esetén a debug esethez hasonlóan illeszkedtek a lineáris modell becsléséhez.

4.4.2. Az órajelek hatása

A 4.3 ábrán a mérési pontokat a két processzor órajelének függvényében ábrázoltam. A mérési pontokhoz 60 MHz-nél nagyobb órajelek tartoznak, ezek esetén a mérés időtartama még elfogadható volt. A mérés során a D3 doménben található memórián keresztül történt a kommunikáció. Szaggatott vonallal a mérési adatokra illesztett lineáris modell alapján számított késleltetés és adatátviteli sebesség látható. Erre a mérési pontok jól illeszkednek. A felületen azok az órajelpárok, amik a rendszerben nem lehetségesek, ugyanis az M4-es mag frekvenciája nem lehet nagyobb, mint az M7-es magé, nincsenek ábrázolva. A későbbi ábrákon a teljes modell alapján becsült felületet ábrázolom a könnyebb olvashatóság miatt.

A mérési pontok jól illeszkednek a becsült felületre, aminek függőleges síkokkal vett metszeteiből látszik, hogy az M7-es mag frekvenciájának változtatásával a kommunikáció

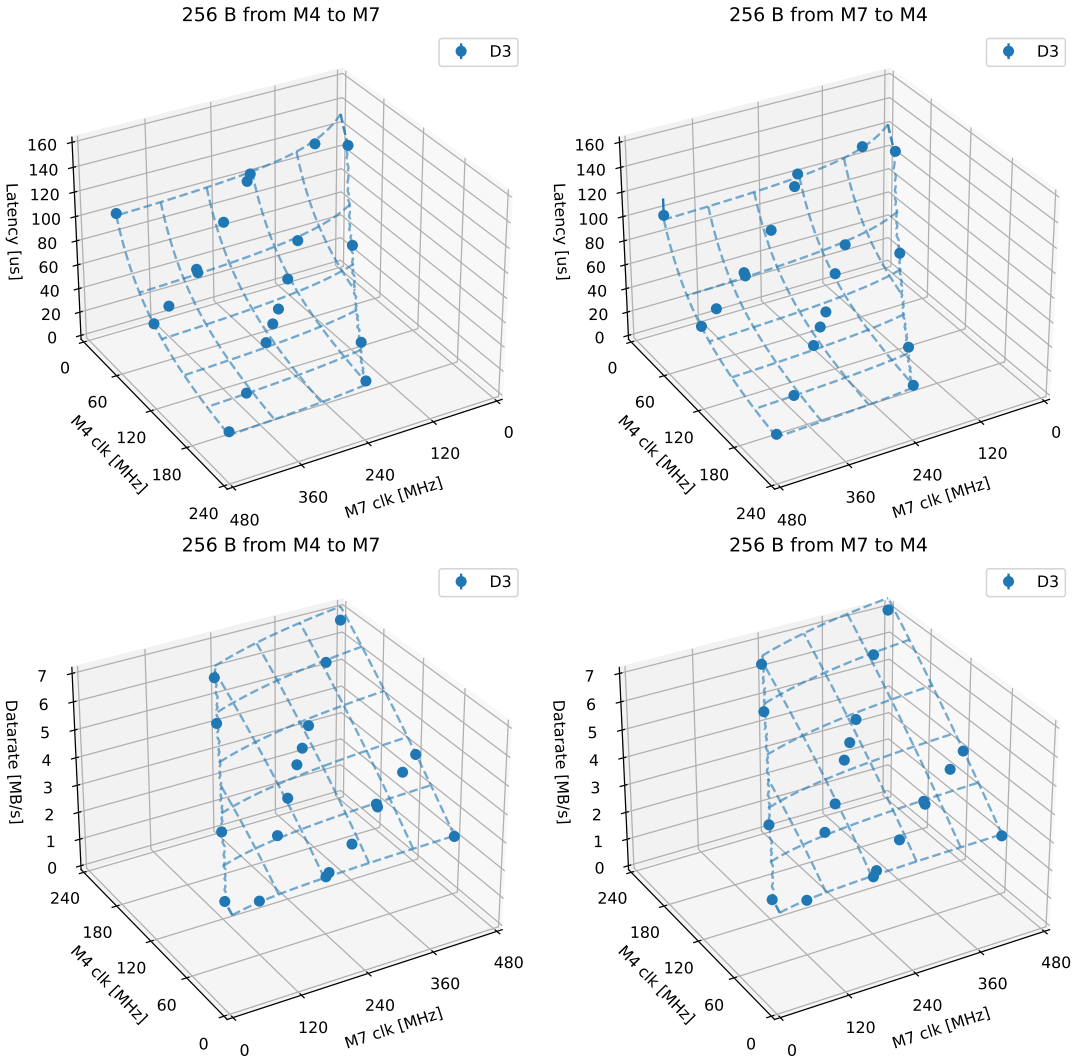


4.2. ábra. Optimalizált fordítás, valamint adatméret hatása a késleltetésre és az adatátviteli sebességre

teljesítménye nem változik jelentősen. Ezzel szemben az M4-es mag órajelfrekvenciáját változtatva a mért értékek sokat változnak, a kommunikáció sebessége erre sokkal érzékenyebb. A két oszlop ábrái nem különböznek látványosan, a küldés iránya nincs jelentősen hatással a teljesítményre.

Az órajelek változtatásának hatását az adatméret függvényében is megfigyelhetjük. A 4.4 ábrán az M7, a 4.5 ábrán pedig az M4 frekvenciáját változtattam. Lászik, hogy az órajelek megváltozása a késleltetés állandó összetevőjét és az adatmérettel arányos részét is megváltoztatja, és az M7-es mag hatása az M4-nél sokkal kisebb.

A lineáris modell szerint az két mag frekvenciájára való érzékenység különbséget magyarázhatjuk azzal, hogy a lassabb processzoron végzett műveletek késleltetése sokkal nagyobb, mint a gyorsabb processzoron. Kód optimalizációjakor a legfontosabb szempont, hogy melyik kódrészlet vesz hosszú időt igénybe, ugyanis annak javításával van lehetőségünk megfelelő gyorsulást elérni. Ehhez hasonlóan itt is a késleltetés jelentős részét adó számításokat gyorsítva tudunk nagy mértékben javítani a kommunikáció teljesítményén, vagyis az M4-es magot felgyorsítva.



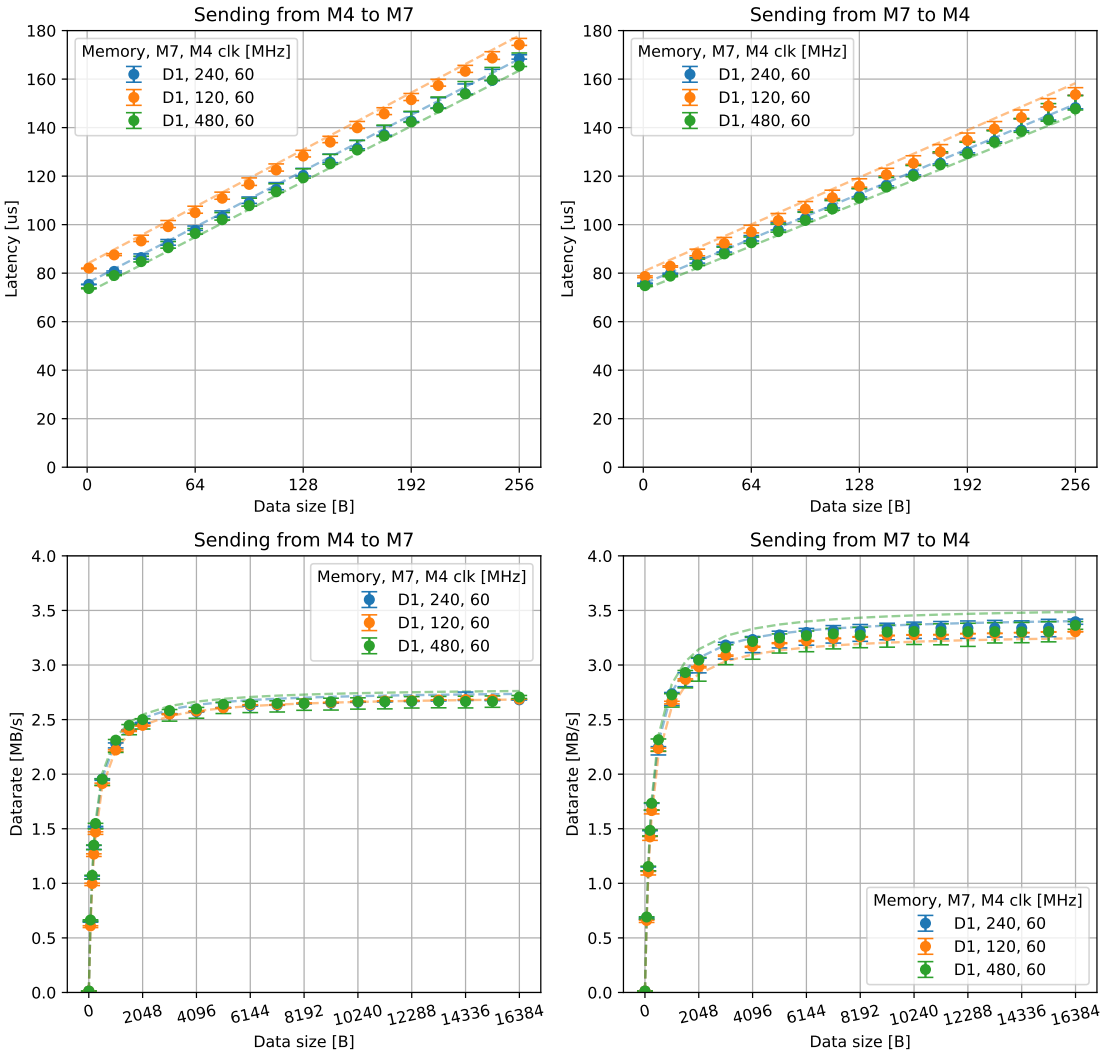
4.3. ábra. Késleltetés és adatátviteli sebesség órajelfüggése

4.4.3. A memóriák közötti különbség

Az 4.6 ábrán bemutatom a rendszerben található három memória közötti különbséget. Az órajelpárok a vízszintes tengelyeken ábrázolt rácspontokra esnek. A felületek alakja az eddigiekhez hasonlóan alakul és a mérési eredményekre jól illeszkedik, tehát a modell mindhárom memória esetén jó közelítés az órajelfüggésre. Látszik, hogy a kommunikáció teljesítményében az iránytól függően van különbség. Ez legjobban a D1 doménben lévő memória késleltetésén megfigyelhető.

A három memória teljesítménye az adatméret függvényében a 4.7. ábra szerint alakul. A kommunikáció adatátviteli sebessége az M4-es magról üzenetet küldve minden memória esetén lassabb, mint fordított irányban. A D2 és D3 doménbeli memória sebessége hasonló, a D1 a leglassabb.

Az irányok közötti különbségből arra következtethetünk, hogy a küldés vagy fogadás több művelettel jár. Emiatt amikor a kisebb műveletigényű fogadás fut a lassabb processzoron, akkor gyorsabb működést kapunk. A sebességet d_{M7} és d_{M4} paraméterek befolyásolják elsősorban. Az M4-es processzoron a d_{M4} paraméter az esetek többségében nagyobb küldéskor, mint fogadáskor, ami nagyobb késleltetést jelent bájtonként, azonban ez nem minden konfigurációban jelenik meg.



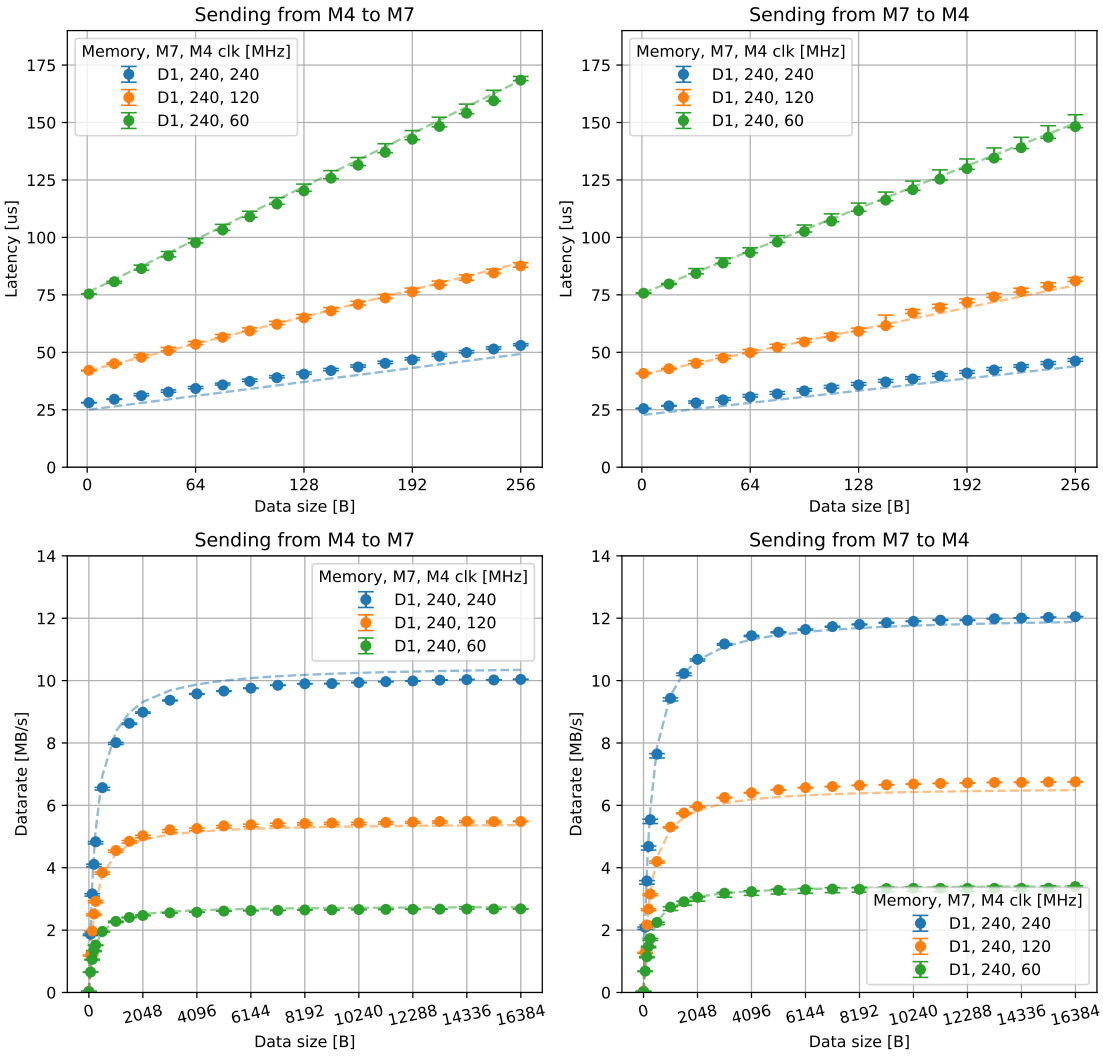
4.4. ábra. Órajelfüggés az M7 órajelének változtatásával az adatméret függvényében

A memóriák közötti sebességkülönbség elsősorban az M4-es magtól függhet, mivel a kommunikáció késleltetésének nagyobb része itt történik. Látszik, hogy a d_{M4} paraméterek jóval nagyobbak, mint a d_{M7}, így azonos órajel mellett is sokkal lassabban képes az adatok másolására.

4.4.4. A gyorsítótár hatása

A 4.8 ábrán látható a bekapcsolt gyorsítótár esetén a memóriák összehasonlítása. A lineáris modell szerinti felülethez igen közel helyezkedik el minden mérési pont. Az órajelpárok a korábbihoz hasonlóan a vízszintes tengelyek szerinti rácson helyezkednek el. A D2 domén memóriájának késleltetése és sávszélessége jelentősen eltér a kommunikáció irányának megváltoztatásával. A D2 doménen keresztül M7-ről M4-re küldött üzenet (jobb alsó ábra) szerint nagy látszik, hogy amikor M4 órajele nagy és megfelelő memórián keresztül kommunikálunk, akkor már az M7 frekvenciájának is arányában komolyabb hatása lehet, ugyanis ilyenkor az M4 késleltetése kisebb arányt képvisel a teljes késleltetésből.

A 4.9 ábra összehasonlít minden memóriakonfigurációt, vagyis a három domént gyorsítótárral és nélküle. A késleltetések ezesetben 512 B-ig vannak ábrázolva, hogy a meredek-

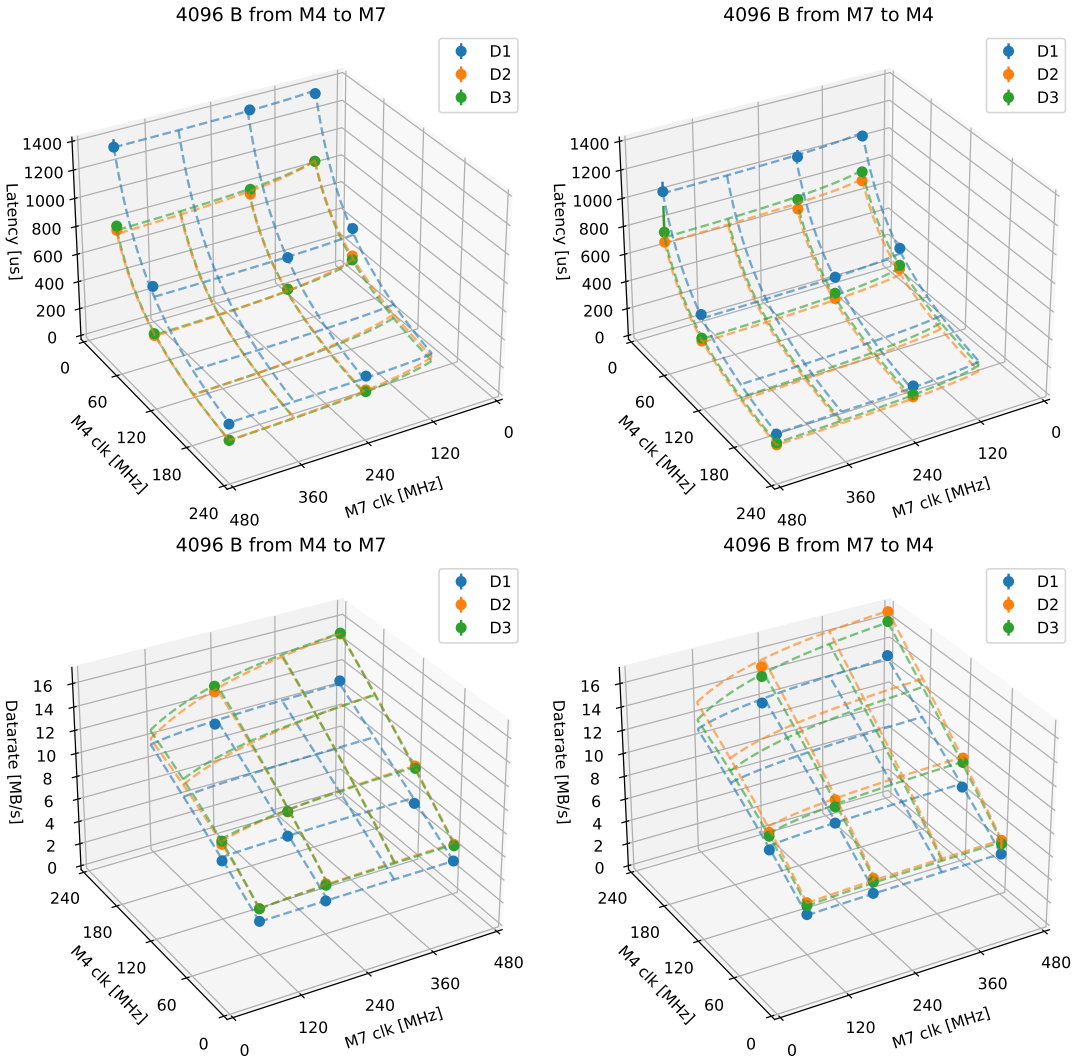


4.5. ábra. Órajelfüggés az M4 órajelének változtatásával az adatméret függvényében

ségek viszonya jobban kivehető legyen. A szaggatott vonallal jelölt, lineáris modell által becsült eredmények nem minden esetben teljesen pontosak, de közel helyesek.

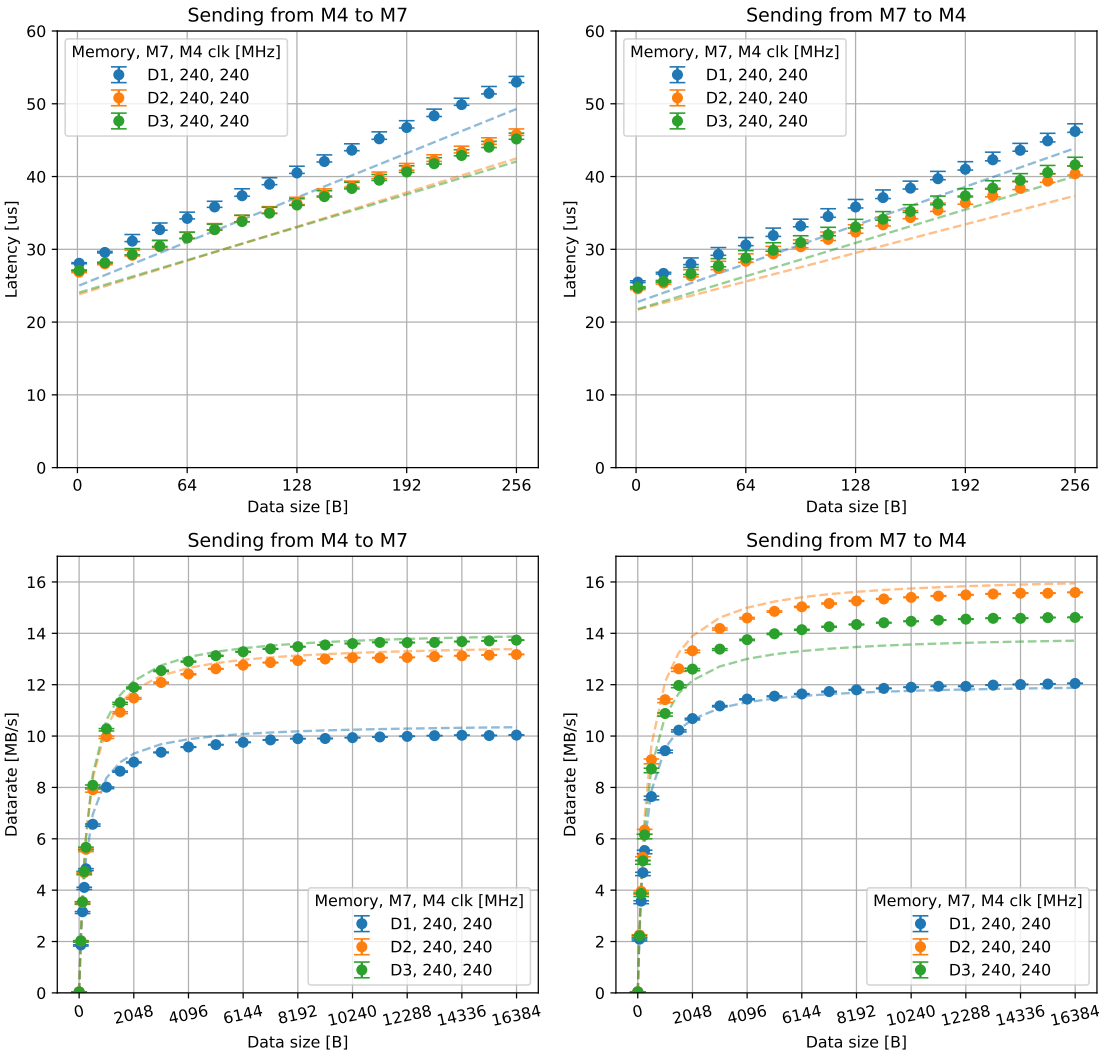
Az M4-M7 irányban a D2 és D3 memória gyorsítótártól függetlenül közel megegyezik (4.8. ábra, 4.6. ábra). A cache bekapcsolása a késleltetést képes csökkenteni mindhárom memória esetén, és ezek ebben az esetben, amikor a két mag órajele megegyezik, akkor a memóriától független. A D1 domén esetén azonban a másolás sebességét a gyorsítótárazásból kizárt memóriából nem befolyásolja, így a legjobb adatátviteli sebességet nem befolyásolja, a hatása a sebességre elhanyagolható. A D2 és D3 memóriák által elérhető sávszélesség gyorsítótár bekapcsolásával kismértékben még csökken is.

Az M7-M4 irányban is megegyezik a késleltetés állandó összetevője minden memória esetén, a gyorsítótár bekapcsolásakor pedig ez az érték csökken. A leggyorsabb memória, amely a maximális órajelek mellett az elérhető legjobb sebességet mutatja, a D2-es doménben található. A gyorsítótár bekapcsolásával nagyon nagy mértékű javulást tudunk elérni az adatátviteli sebességben ennek az IPC-re való felhasználásakor. A D3 domén esetén ebben az irányban is megjelenik a cache adatmásolás lassító hatása, a korábbiaknál hangsúlyosabban is. Ezt a modell paramétereiből is le tudjuk olvasni (4.1. táblázat), például D3 esetén a gyorsítótár bekapcsolásakor megnövekszik a d_{M4} értéke. A D1 domén memóriája ezesetben is a leglassabbnak bizonyult.

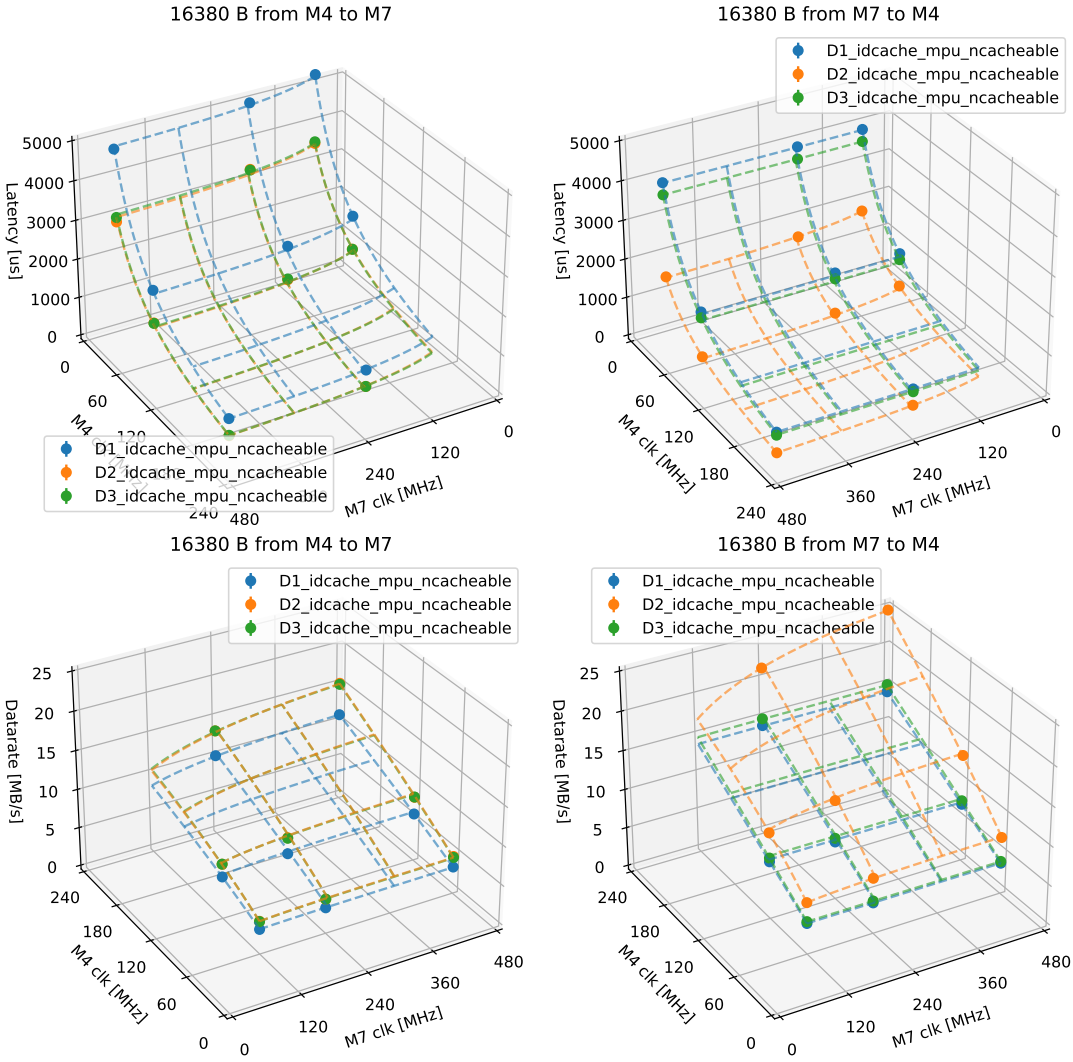


4.6. ábra. A memóriák közötti különbség az órajelek függvényében ábrázolva

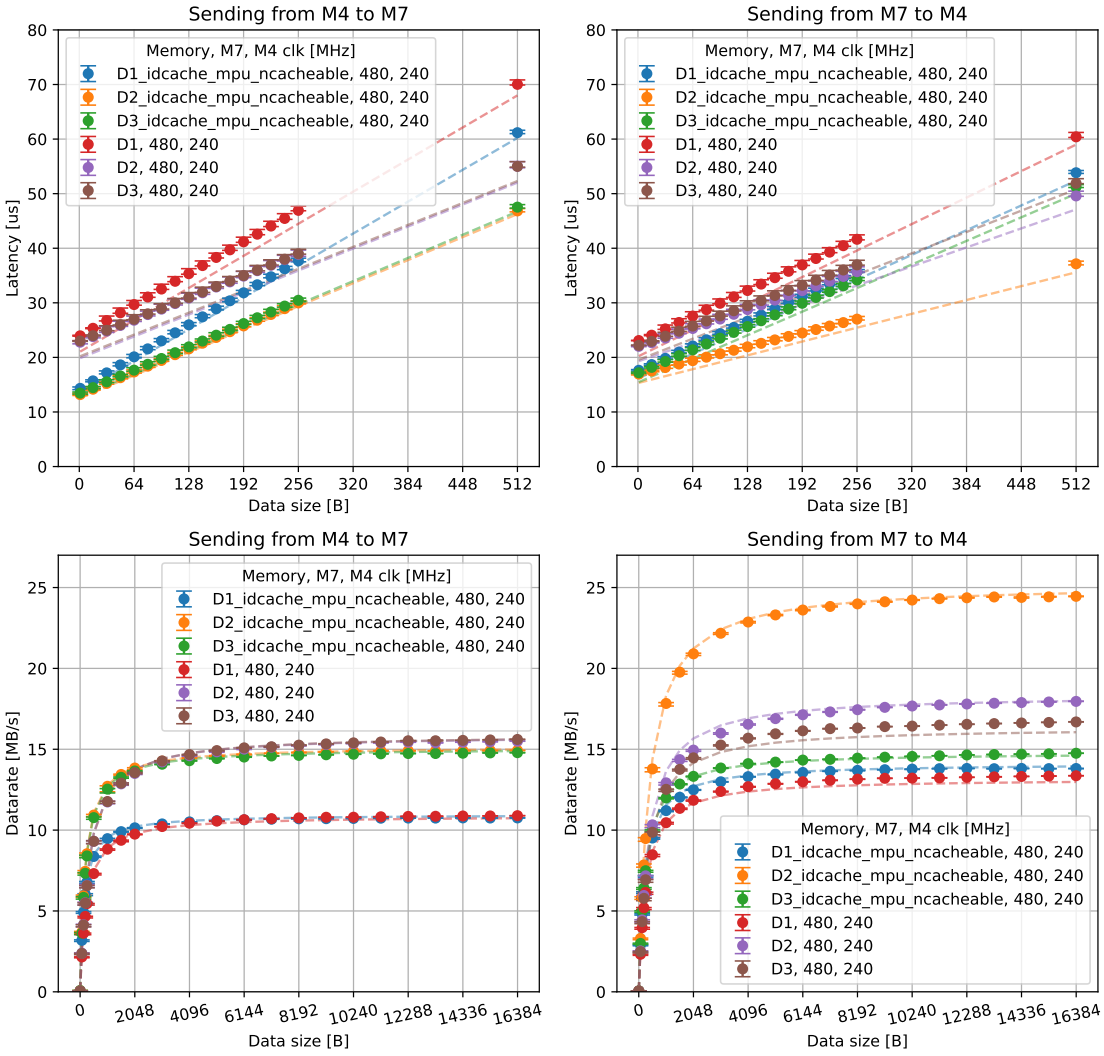
A gyorsítótár lassító hatását a nem cache-elt memóriaterületről való másolásra nehéz megállapítani, ellenőrizve ez a jelenség minden órajelfrekvencia esetén fennáll. A cache gyorsító hatása a késleltetés állandó tagjára könnyen érthető, ilyenkor a műveletek nagy része feltehetően nem a megosztott memória hozzáférése, a gyorsítótárból kis késleltetésű hozzáférés lehetséges a használt adatokhoz.



4.7. ábra. A memóriák közötti különbség az adatméret függvényében ábrázolva



4.8. ábra. A memóriák közötti különbség bekapcsolt gyorsítótárral az órajelek függvényében



4.9. ábra. A memóriák közötti különbség bekapcsolt gyorsítótárral az adatméret függvényében ábrázolva

5. fejezet

Összefoglalás

Megvizsgáltam egy többmagos heterogén architektúrájú mikrokontroller IPC teljesítményét, miközben a processzormagokon külön-külön RTOS futott. Ehhez összeállítottam egy mérési elrendezést a mikrokontrolleren, amit PC-ről Python scriptek vezéreltek. A feldolgozott mérési eredményeket ábrázoltam és kiértékeltem, valamint ezekre egy egyszerű lineáris modellt illesztettem.

Az eredmények alapján lehetséges a mikrokontrolleren egy összetett alkalmazás kialakítása, amiben a mérési eredmények alapján jól indokolt szoftverarchitektúra alakítható ki az IPC teljesítményének figyelembevételével. A lineáris modell felhasználható a kommunikáció várható viselkedésének közelítő becslésére.

A mérési eredmények alapján a D2-es doménben található memória a leggyorsabb mindkét irányú kommunikáció esetén, a D1-ben elhelyezkedő pedig a leglassabb. A kommunikációhoz használt adatméret megválasztásakor a követelmények alapján kell dönteni a megengedhető késleltetésről. Az adatméret növelésével a hasznos sávszélesség növekszik. A pontos adatméret meghatározásában is segítséget nyújt az illesztett lineáris modell. A processzorok órajelének beállításakor a lassabb mag frekvenciájára a kommunikáció teljesítménye igen érzékeny. Az IPC sebességét általánosan is a lassabb processzor paraméterei határozzák meg, az optimalizáció során is ez alapján érhető el jelentős javulás. A gyorsítótár bekapcsolása az esetek legnagyobb részében jelentős gyorsítást jelent a kommunikációban, különösen kisebb adatméreteknél. Az elérhető legnagyobb sávszélességet nagyon nagy méretek esetén azonban kis mértékben csökkentheti is.

Továbbfejlesztési lehetőség az eredmények felhasználása alkalmazásfejlesztéshez. Fontos opció megvizsgálni az optimalizált kód hatását minden konfiguráció esetén. Érdemes a lineáris modell validációja újabb mérések alapján, hogy az általánosítóképességet értékelhessük. A mérések más hardveren is megismételhetőek, a jelenlegi eredmények ezzel összevethetőek.

Köszönetnyilvánítás

A dolgozatban bemutatott munka részleges támogatást kapott a BME-MIT-en keresztül az Európai Unió Horizont 2020 Kutatási és Innovációs Programjából a 872614. számú támogatási szerződés keretében, amelynek címe SMART4ALL: Selfsustained Cross Border Customized Cyberphysical System Experiments for Capacity Building among European Stakeholders.

This work has received partial funding at BME-MIT from the European Union's Horizon 2020 research and innovation program under Grant Agreement No 872614 - SMART4ALL: Selfsustained Cross Border Customized Cyberphysical System Experiments for Capacity Building among European Stakeholders.

Irodalomjegyzék

[1] Gene Amdahl. Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, n.j., apr. 18–20). *Solid-State Circuits Newsletter, IEEE*, 12:19 – 20, 02 2007.

[2] STMicrolectronics. Stm32h7mcus for rich and complex applications. https://www.st.com/resource/en/product_presentation/microcontrollers_stm32h7_series_product_overview.pdf. Elérve: 2023. október.

[3] Gábor Fekete and Tamás Kovácsházy. Execution of resource intensive tasks on a heterogeneous soc for low-latency embedded compute. In *2023 24th International Carpathian Control Conference (ICCC)*, pages 124–129, 2023.

[4] Nik Jedrzejewski. Three reasons why embedded heterogeneous systems are more efficient, 2016. https://www.nxp.com.cn/company/blog/three-reasons-why-embedded-heterogeneous-systems-are-more-efficient:BL-3-REASONS-EMBEDDED-SYSTEMS-EFFICIENT. Elérve: 2023. október.

[5] ARM. Learn the architecture - introducing the arm architecture. ARM Documentation, 2023. https://developer.arm.com/documentation/102404/0201. Elérve: 2023. október.

[6] Nico De Witte, Robbie Vincke, Sille Van Landschoot, Eric Steegmans, and Jeroen Boydens. Evaluation of a dual-core smp and amp architecture based on an embedded case study. *CW Reports*, 2013.

[7] STMicroelectronics. *UM2408: STM32H7 Nucleo-144 boards*, 2022. https://www.st.com/resource/en/user_manual/um2408-stm32h7-nucleo144-boards-mb1363-stmicroelectronics.pdf. Elérve: 2023. október.

[8] S.-L Tsao and S.-Y Lee. Performance evaluation of inter-processor communication for an embedded heterogeneous multi-core processor. *Journal of Information Science and Engineering*, 28:537–554, 05 2012.

[9] Ms Krishnaveni and Ruby Durairaj. Comparing and evaluating the performance of inter process communication models in linux environment. *International Journal of Trend in Research and Development (IJTRD)*, pages 51–55, 10 2016. Special Issue Published in IJTRD.

[10] Hyok-Sung Choi and Hee-Chul Yun. Context switching and ipc performance comparison between uclinux and linux on the arm9 based processor. Technical report, Samsung Electronics, 2004. http://opensrc.sec.samsung.com/document/uc-linux-04_sait.pdf.

[11] FreeRTOS project. *FreeRTOS documentation*. Amazon Web Services. https://www.freertos.org/RTOS.html. Elérve: 2023. október.

[12] Nitin Dahad. embedded survey 2023: more ip reuse as workloads surge, 2023. https://www.embedded.com/embedded-survey-2023-more-ip-reuse-as-workloads-surge/. Elérve: 2023. október.

[13] FreeRTOS project. *Simple Multicore Core to Core Communication Using FreeRTOS Message Buffers*. Amazon Web Services. https://www.freertos.org/2020/02/simple-multicore-core-to-core-communication-using-freertos-message-buffers.html. Elérve: 2023. október.

[14] Tamás Kovácsházy and Gábor Fekete. Application experiment with the standard linux services for asymmetric multiprocessing on heterogeneous system on a chips. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6, 2022.

[15] Anders Ive. Runtime performance evaluation of embedded software. In *8th Nordic Workshop on Programming Enviroment Research*. Citeseer, 1998.

[16] Jay Fenlason and Richard Stallman. *The GNU Profiler*. https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html. Elérve: 2023. október.

[17] Chandler Carruth. Tuning c++: Benchmarks, and cpus, and compilers! oh my!, 2015. Előadás a CppCon keretében: https://youtu.be/nXaxk27zwlk?si=e1u1sjC4uhpCy50q. Elérve: 2023. október.

[18] Matthias Grünewald, Jörg-Christian Niemann, and Ulrich Rückert. A performance evaluation method for optimizing embedded applications. In *Proceedings of the 3rd IEEE International Workshop on System-On-Chip for Real-Time Applications*, pages 10–15, Calgary, Alberta, Canada, 2003.

[19] Bryce Adelstein-Lelbach. Benchmarking c++ code, 2015. Előadás a CppCon keretében: https://youtu.be/zWxSZcpeS8Q?si=E9eo_3DagD-tCor5. Elérve: 2023. október.

[20] STMicroelectronics. *RM0399: STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs*, 2023. https://www.st.com/resource/en/reference_manual/rm0399-stm32h745755-and-stm32h747757-advanced-armbased-32bit-mcus-stmicroelectronics.pdf. Elérve: 2023. október.

[21] STMicroelectronics. *AN5557: STM32H745/755 and STM32H747/757 lines dual-core architecture*, 2022. https://www.st.com/resource/en/application_note/an5557-stm32h745755-and-stm32h747757-lines-dualcore-architecture-stmicroelectronics.pdf. Elérve: 2023. október.

[22] Mikrokontroller projekt forráskódja. https://github.com/kovacsdotgergo/stm32h745-ipc-visu.

[23] Feldolgozó és megjelenítő scriptek forráskódja. https://github.com/kovacsdotgergo/stm32h745-ipc.

[24] STMicroelectronics. *Introduction to memory protection unit management on STM32 MCUs*, 2023. https://www.st.com/resource/en/application_note/an4838-introduction-to-memory-protection-unit-management-on-stm32-mcus-stmicroelectronics.pdf. Elérve: 2023. október.